INTELIGÊNCIA ARTIFICIAL EM BASIC

A inteligência artificial ao alcance dos micros pessoais

Programas em BASIC ilustrando aplicações práticas

O interesse pela Inteligência Artificial (AI) tem sido bastante grande nos últimos tempos, embora esteja ainda limitado o seu uso, não só devido à natureza dos problemas que se procura resolver com a ajuda da AI, mas também à falta de bibliografia que familiarize com o assunto os entusiastas e os estudantes.

Paralelamente à apresentação de uma visão coerente dos diferentes aspectos da AI, este livro pretende mostrar que já são viáveis, atualmente, as suas aplicações práticas. De maneira simples e objetiva o texto vem, assim, esclarecer questões fundamentais, ilustrando os métodos com exemplos de programas.

Todos os programas são escritos em BASIC, de forma a tornar o livro acessível ao maior número possível de leitores e a demonstrar que a Inteligência Artificial está ao alcance dos micros pessoais.

ISBN 85-7001-395-7
(Edição original: ISBN 0-408-01373-7 Newnes Technical Books, England.)

JAMES

INTELIGÊNCIA ARTIFICIAL EM BASIC



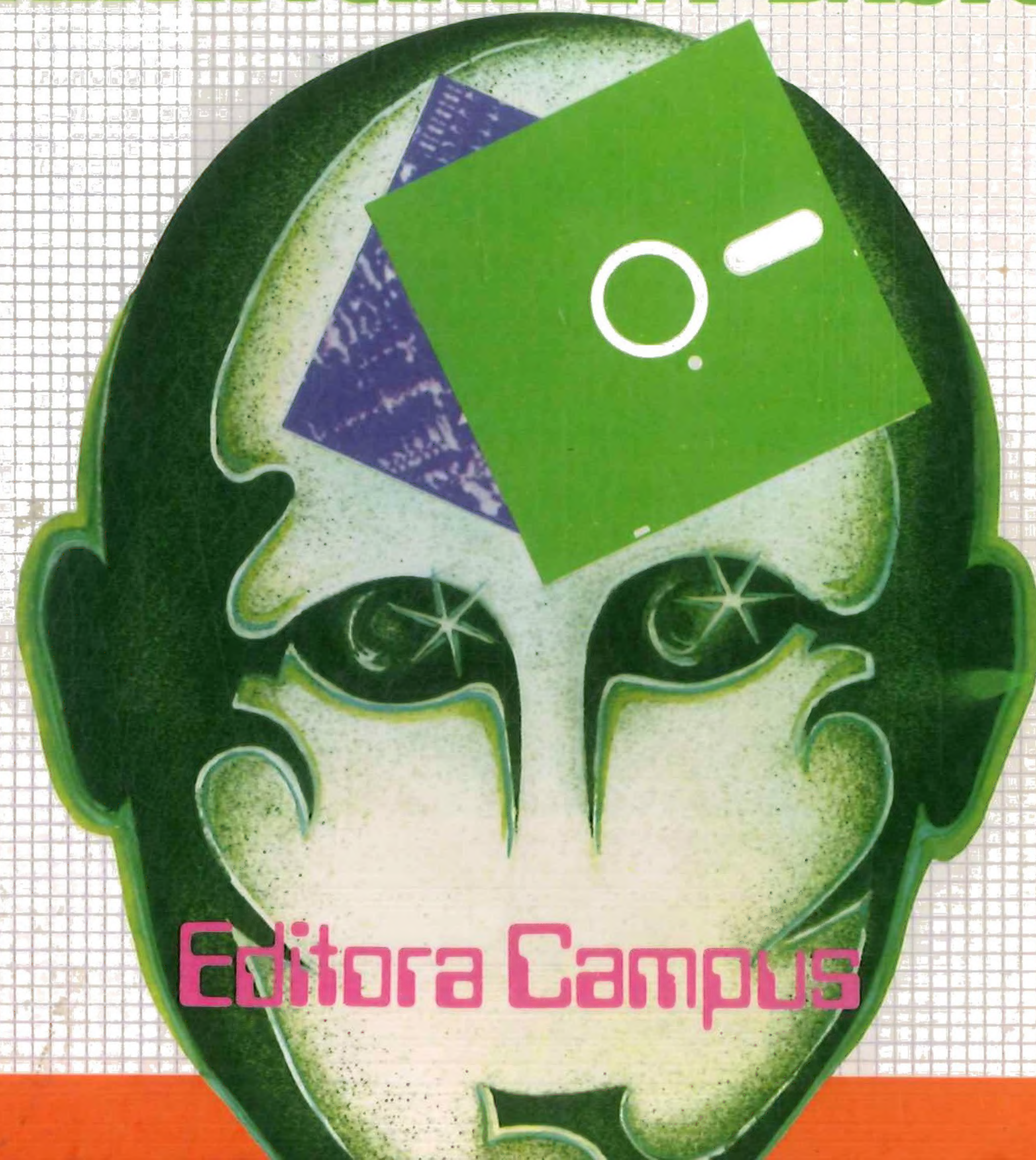
2ª EDIÇÃO

MIKE JAMES

A inteligência artificial ao alcance dos micros pessoais

Programas em BASIC ilustrando aplicações práticas

INTELIGÊNCIA ARTIFICIAL EM BASIC



INTELIGÊNCIA ARTIFICIAL EM BASIC



Consultor Técnico

Solon Benayon da Silva
Engenheiro Eletrônico UFRJ
Analista de Sistemas e Pesquisador
do Instituto Latino-Americano
de Pesquisas e Sistemas da IBM.

INTELIGÊNCIA ARTIFICIAL EM BASIC

MIKE JAMES

2ª edição

TRADUÇÃO
Séfora Junqueira dos Santos
Leopoldo Junqueira dos Santos

REVISÃO TÉCNICA
José Abel Royo dos Santos
Professor do Departamento de Ciências Exatas da Escola Federal de Engenharia de Itajubá – EFEI

Editora Campus Ltda.
Rio de Janeiro

Do original:
Artificial Intelligence in BASIC

Borough Green, Sevenoaks, Kent TN 15 8PH, England.

1986, 2ª Edição.

Capa
Otavio Studart

Projeto Gráfico, Composição e Revisão
Editora Campus Ltda.
Qualidade internacional a serviço do autor e do leitor nacional.
Rua Barão de Itapagipe 55 Rio Comprido
Tel.: (021) 284 8443 Telex (00038) 021-32606
20261 Rio de Janeiro RJ Brasil
Endereço Telegráfico: CAMPUSRIO

ISBN 85-7001-395-7
(Edição original: ISBN 0-408-01373-7 Newnes Technical Books, England.)

Ficha Catalográfica
CIP-Brasil. Catalogação-na-fonte.
Sindicato Nacional dos Editores de Livros, RJ.

J29i
2.ed.

James, Mike
Inteligência artificial em BASIC/ Mike James; tradução Séfora Junqueira dos Santos, Leopoldo Junqueira dos Santos. — 2. ed. — Rio de Janeiro: Campus, 1986.

Tradução de: Artificial intelligence in BASIC.
Bibliografia.
ISBN 85-7001-395-7

1. BASIC (Linguagem de programação para computadores). 2. Inteligência artificial — Processamento eletrônico de dados. I. Título.

86-0977

CDD — 001.6424
CDU — 800.92BASIC

Prefácio

Inteligência Artificial (AI) é um assunto muito interessante. Entretanto, normalmente fala-se mais sobre AI do que se trabalha com ela, já que os problemas que ela procura resolver são geralmente muito difíceis. Além disso, os programas de AI são, usualmente, longos demais para serem apresentados como parte de uma discussão. O resultado é que se encontram descrições de métodos e listagens de resultados, mas o *software* de AI raramente é visto.

Este livro apresenta algumas das idéias centrais de AI, completadas por programas ilustrativos dos métodos. Tudo o que é necessário para estudá-las é o conhecimento de programação BASIC e um computador pessoal que execute programas BASIC.

Ao par da tentativa de apresentação de uma visão coerente de inúmeras diferentes partes da AI, a finalidade principal deste livro é mostrar que já são viáveis, atualmente, aplicações práticas de AI. Experimentando os exemplos de programas e descobrindo o que eles podem fazer com apenas um pequeno esforço de programação, o programador entusiasta deverá ganhar suficiente confiança nos métodos para dedicar tempo ao desenvolvimento de produtos de real interesse.

Este livro começou como uma série de artigos em *Electronics and Computing Monthly*, a cujo editor, Dave Raven, agradeço pelo encorajamento. Gostaria, também, de agradecer

a Steve Oakey, por ampliar meu interesse em AI, e a Vartkes Goetcherian, pela participação em muitas e longas discussões sobre sistemas especialistas.

Este livro é dedicado a meu filho Samuel Michael — meu projeto pessoal em inteligência natural.

M.J.

Sumário

1 Inteligência computacional: fato, ficção e futuro, 10

2 O enfoque heurístico, 21

3 Quando as heurísticas se encontram: a estratégia de competição, 45

4 Pensamento e raciocínio: sistemas especialistas, 59

5 A estrutura da memória, 90

6 Reconhecimento de padrões, 111

7 Linguagem, 133

8 Abordagens à inteligência, 150

Leitura adicional, 163

Índice analítico, 165

1

Inteligência computacional: fato, ficção e futuro

Um tema sempre presente na ficção científica é o desenvolvimento de um computador cuja inteligência rivalize com a humana. Embora isto, hoje em dia, ainda seja ficção científica, pode ser que o deixe de ser a curto prazo. Laboratórios de pesquisa tentam produzir computadores que tenham algumas características humanas: computadores que falem, computadores que vejam, computadores que pensem. De certa forma, é menos importante saber se estes pesquisadores algum dia alcançarão seu objetivo final de igualar, ou talvez ultrapassar, a capacidade mental humana, do que examinar o progresso que estão conseguindo em cada área em separado; pois, embora seja claro que um computador verdadeiramente inteligente exigiria um *hardware* muito sofisticado para conter toda sua programação, alguns dos aspectos menores, mas muito interessantes, da inteligência já podem ser duplicados até em computadores pessoais.

A maior parte da pesquisa que está sendo realizada em Inteligência Artificial, ou AI como é usualmente conhecida (do inglês *Artificial Intelligence*), é um pouco difícil de ser compreendida. Isto não se deve a uma dificuldade intrínseca da AI, mas porque é o resultado de muitos anos de pesquisa de acadêmicos e engenheiros, cujo interesse maior é a comunicação com seus pares, que já conhecem muito sobre o assunto. O principiante interessado fica de fora! A finalidade deste livro é explicar aquelas idéias de AI que podem ser usadas para produzir programas nos microcomputadores atuais e, assim, dar uma introdução à AI tão prática quanto possível. A parte prática do livro toma a forma de programas escritos em BASIC. Estes programas servem simultaneamente para ilustrar os tópicos em discussão e para dar aos leitores um ponto de partida em suas pesquisas. Enquanto é verdade que a maior parte do trabalho mais avançado em AI exige computadores poderosos e muito caros, é bem possível que alguma pesquisa particular de baixo custo possa enfocar alguma área ou detalhe que tenha sido desprezado. AI é desenvolvimento de *software* e, como tal, é um assunto no qual todos os programadores podem tomar parte. É importante que você digite e use os programas-exemplo, pois grande parte do excitamento do uso de um programa de AI não pode ser bem transmitido por uma simples explicação escrita. Além disso, ao usar mesmo os programas mais simples, você poderá avaliar quanta inteligência é possível produzir mesmo com recursos muito limitados.

Conheça o QI do seu computador

Você pode achar difícil de acreditar que o seu pequeno computador caseiro seja capaz de executar programas que reproduzam alguns aspectos do comportamento humano. Afinal de contas, não é verdade que sua máquina não consegue nem executar programas mais comuns com velocidade suficiente? Também não é verdade que ela costuma ficar sem memória suficiente no meio do desenvolvimento de muitos programas?

Entretanto, se você examinar a gama de programas que um computador pessoal pode executar, poderá detectar dois tipos principais. Alguns programas são simples, mas exigem muito da velocidade de cálculo do computador. Por exemplo, jogos tipo "fliperama", com gráficos móveis, são, acredite ou não, muito simples do ponto de vista da lógica, residindo toda a dificuldade de sua produção na sua necessidade de execução *muito* rápida. O outro tipo são de programas difíceis de escrever pela dificuldade de acompanhar o que eles devem fazer, sendo sua velocidade de execução uma preocupação secundária. Por exemplo, um programa para jogar xadrez é um desafio devido à lógica do jogo de xadrez ser muito sofisticada. Se você conseguir desenvolver a lógica de um tal programa, poderá ainda ter de enfrentar problemas de insuficiência de memória e de falta de velocidade de computação, mas estes são

simples de resolver, comparados com o desenvolvimento da lógica. O fato de que existam programas de jogo de xadrez, mesmo para os menores micros, deve fazê-lo parar e pensar sobre a real capacidade de sua humilde máquina! Jogar xadrez é um ato bastante inteligente e seu micro pode realizá-lo!

Examinando esta evidência, somos levados a concluir que os micros atuais têm uma inteligência potencialmente muito maior do que o *software* relativamente simples que eles executam nos levariam a imaginar.

Inteligência comercial

A importância dos programas de AI não passou totalmente despercebida pelos produtores de *software* comercial, e o futuro próximo trará a introdução de um número cada vez maior de programas "inteligentes". A potencialidade dos programas tende a evoluir devido às forças de mercado. Se você for o primeiro a produzir uma peça particular de *software*, esta pode ser muito simples, mas ter muito sucesso, pois muitas pessoas terão de comprá-la. Quando outras companhias de *software* lançarem produtos semelhantes, terão de acrescentar opções extras e melhoramentos, se quiserem que os compradores prefiram seu produto ao original. A uma certa altura, entretanto, o produto alcança um estado de refinamento difícil de aperfeiçoar e, para causar alguma impressão no mercado, o vendedor de *software* terá de pensar em algo novo. Neste caso, novo, usualmente, significa um aumento na inteligência do programa.

Um bom exemplo disto é o campo de processamento da palavra. Os primeiros programas de processamento da palavra permitiam apenas que um texto fosse alterado e formatado. Programas posteriores incluíam refinamentos tais como processamento de notas de rodapé e compilação automática de índice remissivo. Um salto significativo neste fluxo constante de aperfeiçoamento foi a inclusão de programas de teste da grafia e do texto. Estes executam a maior parte da tediosa tarefa de revisão das provas de texto que anteriormente exigiam inteligência humana para sua realização. Atualmente, estes programas estão cada vez mais inteligentes. O anúncio de um programa de revisão diz que ele testa e oferece sugestões sobre a gramática. Aonde é que isto vai parar?

Além de jogar xadrez e rever provas, outras áreas de AI que já estão sendo exploradas comercialmente incluem análise de decisão (programas que auxiliam um operador humano a tomar decisões em situações difíceis), sistemas especialistas (programas que podem resolver problemas em determinadas áreas) e programação (programas tipo THE LAST ONE e PEARL, que geram programas a partir da descrição de um problema). Entretanto, parece provável que haja muitas outras aplicações interessantes de AI que permaneçam inexploradas simplesmente porque as teorias correspondentes continuam escondidas em periódicos acadêmicos ou mesmo porque as pessoas que têm conhecimento dessas teorias não têm interesse em aplicações comerciais. Não há nenhuma dúvida de que AI tornar-se-á cada vez mais importante em todas as áreas de aplicação de computadores.

Inteligência Auxiliada por Computador

Antes de passar a considerar as diferentes áreas de AI que serão cobertas nos capítulos seguintes deste livro, há ainda algo que podemos aprender examinando os programas comentados de leitura de provas, de grafia e de gramática. Há algumas áreas da inteligência humana que *não* podem ser simuladas usando a tecnologia atual (ou mesmo a previsível). Isto não é razão, entretanto, para não explorar o uso de computadores nestas áreas. Muitas pessoas provavelmente reconheceram a necessidade de um programa que verificasse a grafia das palavras, mas devem tê-lo rejeitado porque produzi-lo significaria, efetivamente, ensinar a um computador a grafia de todas as palavras de uma língua. Quem não abandonaria tal projeto mesmo antes de começá-lo? Este impasse somente foi superado quando alguém concluiu que um programa para testar grafia pode ser escrito muito facilmente se levar em conta a existência de um humano para resolver todas as dúvidas que persistirem após a consulta a um dicionário computacional. Assim, o programa de grafia procura cada palavra em seu dicionário interno e, se encontrá-la, continua na próxima palavra. Se a palavra não existir, o programa pára e pergunta ao operador se a grafia está correta. Se estiver, será acrescentada ao dicionário e o operador humano não será mais incomodado com a verificação da grafia da-

quela palavra. Poderia ser dito que o programa "aprende" a grafia das palavras novas.

Observe que a chave para a produção de um programa de grafia é a redução do alvo ambicioso de escrever um programa autônomo, para o alvo alcançável de produzir um programa mais simples, mas igualmente útil, que pede a assistência de um humano. Em AI, freqüentemente a solução completa para um problema é assunto de pesquisa avançada, mas uma solução parcial "com auxílio humano" é muito mais fácil de se conseguir, e produz um programa útil que combina a precisão do computador com a flexibilidade do humano.

Esta idéia da mistura de inteligência computacional e inteligência humana é relativamente nova e pouco explorada. À medida que o tempo passa, podemos esperar ver cada vez mais computadores atuando desta maneira, como "amplificadores de inteligência". Os programas tradicionais capturavam ou resumiam uma pequena parte do conhecimento ou da inteligência do programador e tornavam-na disponível ao usuário, para executar alguma tarefa específica. Neste sentido, os computadores têm sido usados como armazenadores de inteligência de capacidade muito limitada. O que está para ser feito no futuro é desenvolver *software* que capture um tipo de inteligência mais complexa e geral, que possa ser usada como um auxílio ao raciocínio do usuário humano.

O Que é Inteligência?

O problema com o termo AI é que, embora o significado de "artificial" seja bastante claro, é muito difícil ser preciso quanto ao significado de "inteligência". A resposta tradicional dada pelos psicólogos — "inteligência é o que é medido pelos testes de inteligência" — é totalmente inútil na área de inteligência computacional. Se interpretássemos a palavra inteligência desta maneira, toda a pesquisa de inteligência artificial se resumiria em encontrar maneiras de responder às perguntas dos testes de inteligência. O resultado seria um computador que saberia responder a questões intrincadas muito bem, mas não poderia ver, falar, ouvir etc... O problema é que os testes de inteligência admitem a existência prévia de algo chamado "inteligência" — eles medem a inteligência, mas não podem *detectar* sua presença. Ao invés de continuar desta maneira, provocando um fluxo de argumentos filosóficos, é mais razoável descrever brevemente as áreas que são normalmente consideradas partes do campo de estudo de AI, de modo a ilustrar o que pode ser incluído.

Não é fácil categorizar completamente todas as áreas de AI, pois há muita superposição e interação, mas, a fim de simplificar, identificaremos as seguintes áreas:

— visão computacional; produção de fala; reconhecimento de fala/voz; pensamento; raciocínio e resolução de problemas; compreensão e tradução de linguagens.

Permeando todas estas áreas está a distinção fundamental entre um programa que é escrito para resolver um problema e um programa que "aprende". Por exemplo, você poderia escrever um programa que fizesse medidas que identificassem a diferença entre um quadrado e um círculo, ou você poderia escrever um programa que pudesse *aprender* esta diferença. Muitas pessoas acham que os programas que aprendem formam a essência da AI, mas acho que isto ignoraria muitos dos interessantes programas de estratégia fixa. Para evitar ulteriores discussões sobre o que AI é ou não é, consideraremos cada uma das áreas enumeradas acima, e examinaremos o que poderemos esperar conseguir em cada uma delas, com um computador pessoal típico.

Visão e Reconhecimento

Visão computacional é uma importante área de pesquisa, pois o uso crescente de robôs na indústria depende dela. É possível comprar, por uma quantia razoável, pequenos braços robóticos que podem ser ligados à maioria dos micros. Estes braços podem ser usados para mover objetos com pequena dificuldade, desde que se encontre alguma maneira de dizer à máquina onde estão os objetos, já que, sem visão, a máquina fica restrita a movimentos cegos. O problema com a visão computacional é que, em primeiro lugar, ela exige um *hardware* muito caro — uma câmera de TV, no mínimo — e, em segundo lugar, um computador realmente rápido. A quantidade de informação presente numa cena típica é tão grande que mesmo os mais rápidos computadores têm dificuldade em realizar mesmo

as mais simples análises em uma quantidade razoável de tempo.

Infelizmente, visão computacional é uma área que está fora do alcance de todos, exceto, talvez, um pequeno número de entusiastas muito dedicados. Isto não significa que devemos abandonar a área de vez, já que há vários subprodutos interessantes da pesquisa de visão computacional que estão acessíveis a qualquer pequeno computador, mesmo com poucos recursos gráficos. Podemos examinar a maneira pela qual formas simples podem ser reconhecidas e, mesmo, tentar um programa de reconhecimento de letras que pudesse formar uma base para um sistema de leitura se acoplado a algum *hardware* extra. Estes tópicos são tratados no Capítulo Seis.

Produção de Fala

A área de produção ou síntese de fala é uma das áreas de AI de maior sucesso comercial atualmente. Você pode comprar computadores que têm chips especiais que produzem qualquer palavra num vocabulário de aproximadamente 200 palavras. Pelo fato desta solução de AI ter sido implementada via *hardware*, pouco há que fazer via *software* em um micro normal. Se o seu micro não tem síntese de fala, é mais fácil e mais barato comprar o *hardware* necessário do que tentar escrever o seu próprio *software*. Vale a pena examinar os princípios que estão por trás da geração de fala, mas, fora isto, há pouco de prático que se possa fazer. Entretanto, se você já tiver um sintetizador de fala, grande parte do material incluído no Capítulo Sete sobre linguagem e compreensão de linguagem irá ajudá-lo a tirar o máximo dele.

Reconhecimento de Fala e de Voz

Tal como a produção de fala, este tópico é melhor tratado por *hardware* especial. Existem alguns painéis que podem ser adicionados aos computadores caseiros que permitirão a estes reconhecer uma entre um pequeno número de palavras. Devido ao *hardware* extra exigido, este livro não trata diretamente do reconhecimento da fala, mas a maior parte do material que trata de reconhecimento em geral, e o de linguagem em particular, poderá ajudá-lo a compreender e a tirar o melhor proveito de qualquer *hardware* que você resolva comprar.

Compreensão e Tradução de Linguagem

O objetivo da compreensão de linguagem é produzir um programa que mantenha uma conversação razoável com o usuário. Tradução de linguagem pode parecer uma tarefa mais simples, resumindo-se simplesmente ao uso de um dicionário de tradução. De fato, entretanto, boa tradução de linguagem exige muito de compreensão de linguagem, para evitar traduções idiotas de expressões idiomáticas tipo "mantenha seu nariz na pedra de amolar".*

Compreensão completa de linguagem não é possível com um micro, pois ele não teria memória suficiente e, além disso, há também o pequeno detalhe de que ninguém parece saber direito ainda como fazê-lo! Entretanto, estudando o que já se conhece sobre linguagem, é possível produzir alguns programas muito úteis. Em particular, você pode escrever um pequeno programa que possa manter uma conversa limitada com você mesmo, e também analisar material escrito para verificar quão difícil ele é de ler. Além disto, um estudo da gramática de uma língua pode lançar muita luz na área de linguagens computacionais.

Pensamento, Raciocínio e Resolução de Problemas

Esta é a maior área de AI e, na realidade, é um repositório de tudo o que não se enquadre naturalmente nas outras categorias anteriores. Pensar, raciocinar e resolver problemas são coisas que, sem dúvida, os humanos fazem, mas são também, em geral, associadas muito intimamente com outras ações. Por exemplo, recebemos freqüentemente, problemas via linguagem ou visão e isto tende a confundir a ação de Resolução de Problemas. Para estudar estes problemas, é necessário trabalhar em áreas pequenas e bem definidas, que gerem problemas fáceis de especificar para um computador. Uma das primeiras áreas

---

* N.T. Do original em inglês "Reep your nose to the grindstone". Expressão idiomática cuja tradução seria "trabalhe persistente e arduamente".

a ser atacada na resolução de problemas foi a de prova de teoremas matemáticos. Foi escrito um programa que aceita uma afirmativa em matemática e gera uma prova completa. Muitos dos teoremas provados tomariam boa fatia de tempo de um graduado em matemática, de maneira que o programa estava simulando um processo de raciocínio bastante avançado, mas ele não pode gerar nenhum resultado realmente novo e, neste sentido, é um "raciocinador" muito pobre. Apesar de a criatividade ainda estar fora do alcance mesmo dos maiores computadores, pensamento e raciocínio tornaram-se áreas de rápido desenvolvimento em AI que podem ser alcançadas usando-se mesmo os menores micros, como descobriremos no Capítulo Quatro.

Uma das áreas mais bem conhecidas de resolução de problemas é a de jogos. Muito esforço já foi dedicado à imaginação de métodos para fazer computadores jogar um bom xadrez "tipo humano". Na realidade, não é possível criar programas para jogar xadrez usando BASIC, devido às suas exigências de tamanho e rapidez. É uma agradável surpresa, entretanto, o fato de haver um grande número de jogos mais simples que constituem excelentes exemplos de resolução de problemas em computador. Jogar usando computadores constitui uma introdução tão boa às idéias fundamentais de AI, que os próximos dois capítulos serão dedicados a este assunto.

Também incluída em pensamento, raciocínio e resolução de problemas, está a atividade conhecida por *aprendizado*. De fato, existe toda uma matéria dedicada às máquinas que aprendem — cibernética. Mais precisamente, cibernética é o estudo das máquinas que podem se ajustar para adaptar-se ao seu ambiente. Aqui encontramos conceitos familiares aos engenheiros eletrônicos, tais como realimentação e laços de controle. É perfeitamente possível escrever programas em BASIC que aprendam ou que se ajustem aos seus ambientes, e no Capítulo Oito este problema será examinado.

O assunto de aprendizado nos leva à área de memória. A memória humana é muito diferente da Memória de Acesso Aleatório normal (RAM) que é encontrada dentro de todo computador. De uma certa forma, a memória humana é muito desorganizada: se você tenta se lembrar de alguma coisa, existe uma grande chance de que você recorde também uma porção

de outras coisas relacionadas a ela. Diz-se que a memória humana é "associativa" devido ao fato de as coisas não serem simplesmente memorizadas, mas serem memorizadas relacionadas a outras coisas associadas a elas. É possível escrever um programa de simulação da maneira que a memória funciona em um computador bastante pequeno. Veremos como fazê-lo no Capítulo Cinco.

Computabilidade

Computabilidade é o nome de um novo ramo da matemática que tenta descobrir o que os computadores podem e o que eles não podem fazer. Procurando encontrar as limitações teóricas dos computadores, poderemos um dia responder à questão fundamental: é o cérebro humano simplesmente um computador muito sofisticado ou contém ele algum princípio ainda não descoberto? Se não formos mais do que computadores (esta é minha opinião), então, os limites encontrados para os computadores aplicar-se-ão também a nós. Computabilidade é o único assunto realmente não-prático considerado neste livro em algum detalhe (veja o Capítulo Oito), mas é incluída devido a oferecer uma visão fascinante de como raciocinamos acerca do mundo.

De volta ao BASIC

Como já foi dito, os exemplos dados neste livro serão escritos em BASIC. BASIC não é realmente a melhor linguagem para se escrever programas de AI, mas, como é a mais popular linguagem dos micros, isto supera a inconveniência de usá-la. Entretanto, o uso de uma linguagem tal como o BASIC prova, pelo menos, que as técnicas de AI são métodos gerais de programação e não dependem de nenhuma facilidade especial encontrada em linguagens tais como Lisp e PROLOG, que são mais freqüentemente usadas em AI. O dialeto do BASIC usado é o Microsoft BASIC padrão e, como não são usados recursos especiais de gráficos etc..., os programas devem ser facilmente convertidos para qualquer máquina. De fato, na maioria das máquinas, não será necessária nenhuma adaptação. Você pode, entretanto, se quiser, incluir recursos gráficos e sonoros para tornar os programas mais interessantes. Como

muitos dos programas serão relacionados com a implementação de métodos avançados de AI, serão introduzidas técnicas avançadas de BASIC. Com isto, você poderá tambem aprender várias maneiras úteis de usar BASIC, ao estudar os programas.

2
O enfoque heurístico

Para ser capaz de escrever um programa que resolva um problema, é preciso saber como você faria para resolvê-lo. Qualquer um que tenha ao menos um pequeno conhecimento prático sobre computadores, sabe que não há qualquer maneira de conseguir que um computador responda a uma pergunta, simplesmente perguntando; alguém, em algum tempo, tem que ter "programado" a resposta. A dificuldade se encontra no fato de existirem muitos problemas para os quais as pessoas não sabem a solução, e, naturalmente, estes são exatamente os problemas que gostaríamos que os computadores resolvessem. O que é mais surpreendente é ser muitas vezes possível uma pessoa resolver um problema sem saber *como* ele foi resolvido. Mas, para poder escrever um programa que solucione um problema, não é necessário somente ser capaz de solucioná-lo, mas, também, ser capaz de descrever como se chega à solução. A razão para isto é que o programa que você está tentando escrever *é*, na realidade, uma descrição de como você resolveu o problema. Um enfoque à AI argumenta que não importa se não sabemos como fazer para resolver um problema específico, pois o importante é escrever programas que possam solucionar problemas em geral, da mesma forma que as pessoas fazem. Seguindo este enfoque, você poderia chegar a um programa que fosse um solucionador geral de problemas e pudesse resolver problemas específicos, mas talvez não fosse capaz de "contar" como ele o fez! Esta idéia parece muito promissora, mas muito pouco progresso foi feito até agora. Uma

alternativa simples é insistir na tentativa de escrever programas que solucionem problemas específicos, muito embora o método completo de solução possa não ser conhecido.

Solucionando problemas

Considere o problema de ganhar em jogos de xadrez ou damas. Existem algumas pessoas que solucionam este problema muito bem. Elas tendem a ganhar da maior parte de seus adversários e isto sugere que têm um método de jogo que é uma boa solução para o problema. Se você não pudesse encontrar tais peritos, então você poderia concluir que não houvesse solução para o problema xadrez/damas. Em outras palavras, você poderia supor que os jogadores fizessem os movimentos por uma grande variedade de razões, nenhuma das quais tendo relação com a estratégia do jogo, e que o vencedor o fosse, puramente, ao acaso.

A existência de pessoas que se desempenham muito melhor do que iniciantes o fazem, e o sentimento interno de "resolver as coisas", quando se está jogando tal jogo, sugere que as pessoas realmente têm um "programa" dentro de suas cabeças, que soluciona os problemas que estes jogos propõem. Mas, se você tentar explicar como joga xadrez ou damas, provavelmente falhará! Você talvez seja capaz de fazer vagas generalizações ou dar justificativas muito complicadas para um movimento particular, mas, se tentar escrever um programa que jogue no seu lugar, você logo descobrirá como seu conhecimento é pobre.

Para alguns jogos, é possível encontrar umas poucas regras simples que, se seguidas, garantem que você vencerá, ou pelo menos, empatará. Por exemplo, o jogo da velha pode ser resumido desta forma, e uma vez que alguém descubra as regras, ele perde a graça. (Ainda assim, tentar escrever programas que usem as regras para ganhar sempre, ou pelo menos empatar, é um desafio interessante.) Jogos como xadrez e damas, no entanto, são tão complexos que, até agora, ninguém conseguiu encontrar um conjunto de regras que garanta uma vitória. Pode ser que isto seja impossível, ou, talvez, alguém descubra o que nos tem faltado, e reduza os jogos de xadrez e damas ao nível do jogo da velha!

A situação atual é que existem muitos problemas que as pessoas solucionam, e que são muito difíceis, se não impossíveis de serem reduzidos à aplicação de um conjunto de regras que garantam sua solução.

Heurística

Tradicionalmente, um programa é uma lista de instruções para dar uma solução segura para um problema, ou dizer que não existe solução. Tal lista de instruções é chamada *algoritmo,* e os algoritmos formam, até agora, a base fundamental da ciência dos computadores e da programação. No entanto, como já foi mencionado, existem muitos problemas para os quais nós não encontramos soluções algorítmicas, e, se queremos desenvolver a ciência dos computadores, devemos conseguir progressos nestas áreas. Se você examinar a maneira pela qual soluciona problemas, talvez note que o que freqüentemente faz, não é usar um algoritmo, mas aplicar uma coleção desconexa de regras que "parecem" funcionar. Por exemplo, em xadrez você pode seguir a regra "controlar o meio do tabuleiro" e, embora esta, e outras regras como esta, não possam garantir que você pode seguir a regra "controlar o meio do tabuleiro" e, emmentam as chances de você chegar mais perto de uma. Uma regra que crie a possibilidade de se chegar mais perto de uma solução é conhecida como uma *heurística,* e apesar de poder parecer que uma heurística seja um algoritmo de "segunda classe", isto está longe da verdade. Heurísticas não são capazes de garantir uma solução para um problema e podem não lhe dizer quando a solução não existe, mas podem ser usadas para uma grande variedade de situações. Além disto, quando elas conduzirem a uma solução, isto se dará em muito menos tempo do que um algoritmo gastaria para o mesmo problema. A ciência dos computadores e a programação futuras estarão, certamente, mais envolvidas com enfoques heurísticos (ou que combinem heurística e algoritmo) para a solução de problemas.

Heurística computacional e heurística humana

Encontrar uma heurística ainda parece uma tarefa muito difícil, e talvez nós não estejamos nos saindo muito bem nisto, ainda.

O tipo de heurística que as pessoas utilizam é normalmente difícil de descobrir e de expressar, mas isto não deve nos preocupar muito. Estamos tentando encontrar heurísticas que sejam efetivas quando executadas por um computador, e computadores trabalham muito depressa. Sendo assim, é mais fácil encontrar heurísticas simples, e deixar que os computadores as apliquem repetidas vezes, ou de formas muito inteligentes. Em geral, não é exatamente a qualidade da heurística que importa, mas sim a maneira como o computador a usa. Você poderia dizer que uma heurística simples, aplicada muitas vezes, será, provavelmente, tão boa quanto uma heurística complexa aplicada poucas vezes. No entanto, tais generalizações são perigosas, e veremos que, algumas vezes, é importante ter o cuidado de utilizar uma heurística por completo.

No resto deste capítulo, e no próximo, a idéia de uma heurística será explorada através de jogos. No entanto, enquanto os jogos são exemplos excelentes, é importante lembrar que as heurísticas são capazes de enfrentar problemas sérios. Na verdade, existem certas áreas de problemas para os quais, pelo menos atualmente, os algoritmos são incapazes de fornecer quaisquer soluções, e só é possível uma abordagem destes problemas através de heurísticas. Tais áreas incluem toda a gama de problemas tais como: fixação de horários e planejamento de rotas, decisões de como extrair pedaços de formas definidas de tecidos ou outros materiais, com um desperdício mínimo.

O Jogo dos Quadradinhos

O jogo dos quadradinhos é um material excelente para experimentar muitas idéias da inteligência artificial. Em particular, ele demonstra muitos dos elementos usados para implementar um jogo em um computador, tem uma tabela de condições, um conjunto de regras que regem o que constitui um movimento legal, e é difícil encontrar um algoritmo para solucioná-lo. A maior parte das pessoas já estará familiarizada com o jogo dos quadradinhos, de um tipo ou de outro. A versão mais básica do jogo, que é a que usaremos, toma 8 quadrados, numerados de

1 a 8, e os coloca juntos ao acaso formando um quadrado, e deixando um espaço vazio. Por exemplo:

3	4	2
1	5	
6	7	8

O objetivo do jogo é deslocar os quadrados para o espaço vazio, mantendo a forma quadrada, e chegar a um arranjo como:

1	2	3
4	5	6
7	8	

Antes de continuar lendo, tente jogar sozinho, e investigue as razões que o levam a fazer um movimento. Quando criança, eu tive uma versão deste quebra-cabeça, que consistia numa pequena caixa plástica, com os quadrados encaixados uns nos outros, de forma que se pudesse deslizá-los pela caixa, mas não retirá-los, ou roubar!

No caso, como eu, você não tem mais tal brinquedo, e, assim, o programa a seguir permitirá que você jogue através de seu micro. Ele está escrito em uma versão bastante restrita do Microsoft BASIC, assim, você não deverá ter dificuldade em executá-lo em sua máquina.

```
 10 K=0
 20 DIM B(3,3)
 30 FOR I=1 TO 3
 40 FOR J=1 TO 3
 50 K=K+1
 60 B(I,J)=K
 70 NEXT J
 80 NEXT I

 90 I=3
100 J=3
110 FOR K=1 TO 20
120 X=INT(RND(0)*3)-1
130 Y=INT(RND(0)*3)-1
140 IF I+X>3 OR I+X<1 THEN GOTO 120
150 IF J+Y>3 OR J+Y<1 THEN GOTO 120
155 IF ABS(X)+ABS(Y)>1 OR X+Y=0 THEN GOTO 120
```

```
160 T=B(I,J)
170 B(I,J)=B(I+X,J+Y)
180 B(I+X,J+Y)=T
190 I=I+X
200 J=J+Y
210 NEXT K

220 M=1
230 K=0
240 S=0
250 FOR I=1 TO 3
260 FOR J=1 TO 3
270 K=K+1
280 IF B(I,J)<>K THEN S=1
290 IF B(I,J)=9 THEN PRINT "  ";:X=I:Y=J
300 IF B(I,J)<>9 THEN PRINT B(I,J);" ";
310 NEXT J
320 PRINT
330 NEXT I
340 IF S=0 THEN PRINT "SOLUCIONADO":STOP
350 PRINT "MOV. ";M
360 PRINT "MOV. QUAL QUAD.";
370 INPUT N
380 IF N<1 OR N>8 THEN GOTO 350
390 FOR I=1 TO 3
400 FOR J=1 TO 3
410 IF B(I,J)=N THEN A=I:B=J
420 NEXT J
430 NEXT I
440 IF ABS(X-A)+ABS(Y-B)>1 THEN GOTO 350
450 T=B(X,Y)
460 B(X,Y)=B(A,B)
470 B(A,B)=T
480 M=M+1
490 GOTO 230
```

Um Programa para Solucionar o Problema dos Quadradinhos

Se você observar a listagem acima do programa automático dos quadradinhos, deve ser capaz de ver que ele é constituído de uma série de sub-rotinas. Cada sub-rotina executa uma parte diferente do jogo, e o programa não é realmente um único programa, mas uma coleção de sub-rotinas, que podem ser usadas de formas diferentes para explorar o problema. Cada sub-rotina será explicada com detalhes suficientes para que você possa substituí-la por uma versão melhor, ou simplesmente diferente.

O programa está escrito em um Microsoft BASIC padrão, e não deve apresentar problemas de conversão. Algumas declarações que talvez precisem de mudanças, antes que você execute o programa em seu micro, são aquelas referentes ao *array string* M$. Por exemplo, o ZX-BASIC exige que a linha 30 seja trocada por:

30 DIM M$(9,4)

e a linha 6010 por

6010 J = VAL (M$(P,I))

A função RND(0) é usada para produzir números aleatórios variando de 0 a 1, e a função VAL(S$) converterá um número da forma STRING (S$) para a forma numérica.

```
  10 REM PROGRAMA AUTOMATICO DOS QUADRADINHOS
  20 DIM B(9)
  30 DIM M$(9)
  40 DIM M(9)
  50 M=0
  60 Q=0
  70 GOSUB 3000
  80 GOSUB 1000
  90 GOSUB 2000
 100 GOSUB 4000
 110 GOSUB 5000
 120 IF S=0 THEN GOTO 180
 130 GOSUB 6000
 140 GOSUB 7000
 150 M=M+1
 160 GOSUB 2000
 170 GOTO 110
 180 PRINT:PRINT
 190 PRINT "SOLUC. EM ";M
 200 END

1000 FOR I=1 TO 9
1010 B(I)=I
1020 NEXT I
1030 P=9
1040 N=INT(RND(0)*10)+25
1050 FOR Z=1 TO N
1060 GOSUB 6000
1070 GOSUB 7000
1080 NEXT Z
1090 Q=0
1100 RETURN
```

```
2000 PRINT
2010 FOR I=1 TO 9
2020 PRINT B(I);" ";
2030 IF I=INT(I/3)*3 THEN PRINT
2040 NEXT I
2050 RETURN

3000 M$(1)="24  "
3010 M$(2)="135 "
3020 M$(3)="26  "
3030 M$(4)="157 "
3040 M$(5)="2468"
3050 M$(6)="359 "
3060 M$(7)="48  "
3070 M$(8)="579 "
3080 M$(9)="68  "
3090 M(1)=2
3100 M(2)=3
3110 M(3)=2
3120 M(4)=3
3130 M(5)=4
3140 M(6)=3
3150 M(7)=2
3160 M(8)=3
3170 M(9)=2
3180 RETURN

4000 FOR I=1 TO 9
4010 IF B(I)=9 THEN P=I
4020 NEXT I
4030 RETURN

5000 S=0
5010 FOR I=1 TO 9
5020 IF I<>B(I) THEN S=1
5030 NEXT I
5040 RETURN

6000 I=INT(RND(0)*M(P))+1
6010 J=VAL(MID$(M$(P),I,1))
6020 RETURN

7000 T=B(P)
7010 B(P)=B(J)
7020 B(J)=T
7030 Q=P
7040 P=J
7050 RETURN
```


Antes de começar a escrever qualquer programa de jogo, você tem que decidir como vai representar o "tabuleiro". Para o jogo dos quadradinhos, é mais fácil ignorar o fato de que o tabuleiro é bidimensional, e utilizar o *array* M com nove elementos, um para cada posição que um quadrado pode ocupar. Cada posição é numerada de forma que, quando o arranjo final for alcançado, o quadrado 1 estará em M(1), 2 em (M2), e assim por diante. O espaço vazio pode ser representado por qualquer símbolo conveniente, mas as coisas se tornam mais fáceis se for um número, e 9 é a escolha óbvia. Para saber qual quadrado está na posição 1, simplesmente verifique M(1).

Usando esta representação, é fácil imprimir o tabuleiro (sub-rotina 2000), e verificar se a posição final vencedora já foi alcançada (sub-rotina 5000). No entanto, não é tão fácil estabelecer uma posição inicial. O problema é colocar os quadrados em uma ordem aleatória, de uma forma que pudesse ter sido alcançada por movimentos legais dos mesmos. Se você simplesmente gerar uma ordem completamente aleatória e sem restrições para os quadrados, você pode produzir um arranjo inicial que não possa ser transformado no arranjo final sem que, efetivamente, se levante um dos quadrados. (A razão para isto é existirem duas versões do jogo, uma direita e uma esquerda, que não podem ser convertidas, uma na outra, somente com movimentos legais!) A solução é colocar o tabuleiro em sua posição final, e então mover os quadrados através de uma seqüência aleatória de movimentos legais. A sub-rotina 1000 estabelece um tabuleiro aleatório embaralhando um tabuleiro previamente estabelecido na posição final. Isto tem a vantagem adicional de a dificuldade do problema poder ser controlada, pelo grau de embaralhamento. Quanto mais o tabuleiro é embaralhado, mais longe ele fica da posição final, e, portanto, mais difícil deveria ser a solução do problema.

O último problema é detectar os movimentos ilegais. A maneira melhor e mais rápida de fazê-lo é estabelecer uma tabela que liste todos os movimentos possíveis para qualquer posição do "espaço" (o 9 nesta representação). Se você observar a sub-rotina 3000, verá que o *array string M$* é introduzido contendo listas de movimentos legais. Por exemplo, se o "espaço" está na posição 3 do tabuleiro, então M$(3) nos dará o *string*

"26", que significa que você pode mover o quadrado que está na posição 2, ou o quadrado na posição 6, para o "espaço", mas nenhum outro movimento é legal. Esta fácil representação de movimentos legais para qualquer posição do "espaço" é a principal razão pela qual o jogo dos quadradinhos pode ser programado eficientemente em BASIC.

A única outra sub-rotina que merece ser mencionada neste estágio é a sub-rotina 7000, que fará com que cada movimento seja especificado por P, posição atual do "espaço", e J, posição do quadrado a ser movido para o "espaço". Obviamente, isto é somente uma permutação entre M(P) e M(J), e a nova posição do "espaço" é J, que pode ser colocado em P para manter a posição presente atualizada.

Pesquisando uma Solução

Usando somente três sub-rotinas, é possível escrever um programa básico para o jogo dos quadradinhos. Este programa básico está contido no programa listado. Inicialmente, um contador de movimentos M assume seu valor inicial, e, então, são chamadas as sub-rotinas 3000, 1000, 2000 e 4000. Esta parte do programa é sempre a mesma e simplesmente estabelece a tabela de movimento (3000), dá valores iniciais ao tabuleiro (1000) e então o imprime (2000). A sub-rotina 4000 ainda não foi discutida, mas seu papel é colocar a posição do "espaço" em P, depois que o tabuleiro houver sido misturado. Isto só tem que ser feito uma vez porque a posição do "espaço" é acompanhada pela sub-rotina de movimento, 7000, e está sempre em P. A tentativa de encontrar a solução começa com uma verificação, para descobrir se a solução já foi encontrada, com a chamada da sub-rotina 5000. Se não foi, isto é, S<>0, então a única coisa a fazer é um movimento. Neste primeiro programa simples, o movimento a ser feito é escolhido aleatoriamente dentre os movimentos legais pela sub-rotina 6000. Em termos gerais, a sub-rotina 6000 atua como um "gerador de movimentos". Usando a tabela de movimentos legais, M$, é fácil escolher aleatoriamente um movimento legal. Se existem três movimentos possíveis, então tudo que temos a fazer é gerar um número entre um e três, e extrair o caráter correspondente do *string* M$(P). O número de movimentos legais para a posição P é

guardado em M(P), a linha 6000 gera o número aleatório entre 1 e M(P) e a linha 6010 pega o carácter correto de M$(P). Este movimento aleatório é então executado, chamando a sub-rotina 7000, o tabuleiro é impresso e então todo o ciclo é repetido, começando pela verificação para ver se a posição final foi alcançada.

Você poderia pensar que um programa simples como este pudesse ser solucionado pelo computador, usando movimentos aleatórios, em um tempo realmente curto. Afinal de contas, o computador trabalha tão depressa que pode perfeitamente fazer centenas de movimentos errados antes de, finalmente, fazer um certo. Na verdade, apesar de o computador poder fazer muitos movimetos aleatórios rapidamente, ele ainda leva muito tempo para achar a solução. Estou para ver este método simples encontrar uma solução, mesmo depois de estar trabalhando por 24 horas e de ter executado mais de cem mil movimentos!

Uma Heurística para o Jogo dos Quadradinhos

Se você observar os movimentos aleatórios do programa anterior, não poderá deixar de se desesperar ao ver que ele falha em tomar mesmo o mais óbvio movimento que melhoraria a situação. Mesmo que, por algum lapso do destino, ele chegasse a um movimento correto, seria muito provável que, em seguida, ele tomasse a opção errada e começasse a atrapalhar tudo de novo! O observador chega rapidamente à conclusão de que deve haver um enfoque melhor para o problema, mesmo não havendo qualquer algoritmo para sua solução.

Se quiséssemos ter uma medida de quão longe o arranjo atual está do arranjo final, poderíamos usar uma heurística realmente óbvia e escolher o movimento que leva o arranjo o mais perto do desejado.

O único problema é decidir o que queremos dizer por "mais perto". Se você observar a posição atual de qualquer quadrado, poderá descrever a que distância ele está de sua posição final, contando quantos movimentos horizontais e verticais teriam que ser executados para que ele alcançasse a posição de-

sejada, ignorando o "espaço". Por exemplo, o 6, no arranjo a seguir, poderia ser levado para sua posição final, executando um movimento para baixo e dois movimentos para a direita.

6	3	1
2	4	8
	5	7

Assim, ele está a três movimentos de "casa". Observe que não importa que você não pudesse movimentar o 6 por este caminho, porque o "espaço" está no canto inferior esquerdo. Só estamos interessados em usar isto como uma medida aproximada da distância que separa o 6 de sua posição final. Esta medida de quão longe um quadrado está de sua posição tinal é conhecida como "distância em quarteirões", por ser desta forma que a distância entre dois pontos é medida, se você tem que andar através de ruas que se cruzam sob ângulos retos. Assim, para calcular a distância que o tabuleiro está de sua posição final, poderíamos calcular a distância de cada quadrado de sua posição final, e então somá-las, chegando ao número total de movimentos.

Assim, agora temos um significado possível para a expressão "mais perto", em nossa heurística. Examinamos cada um dos movimentos possíveis, e calculamos a distância entre o arrajo final e o arranjo que resultaria se executássemos cada um deles. Obviamente, deveríamos tomar o movimento que nos levaria mais perto do arranjo final. Na prática, calcular a distância do tabuleiro todo pode tomar muito tempo, e podemos simplificar as coisas observando que o movimento que escolheríamos é aquele que produz a maior mudança na distância para o arranjo final. Isto é, não estamos interessados no valor real da distância, mas sim, em diminuí-la. O resultado prático desta observação é que, como somente podemos mover um quadrado de cada vez, é fácil encontrar a mudança produzida na distância. Ela é, simplesmente, a mudança na distância do quadrado que está sendo movido, de sua posição final.

Nossa versão final e prática da heurística é assim como segue:

(a) Para cada movimento possível

(1) calcular a distância atual, do quadrado a ser movido, de sua posição final;

(2) calcular a nova distância que o quadrado estaria de sua posição final, se ele *fosse* movido;

(3) a diferença entre (1) e (2) dá a mudança na distância total, se o quadrado fosse movido.

(b) Escolher o movimento que produzir a maior mudança na distância para o arranjo final.

A alteração do programa dos quadradinhos para adotar esta heurística exige que se escrevam duas novas sub-rotinas (8000 e 9000) para substituir o gerador de movimentos aleatórios na sub-rotina 6000. Para executar a nova versão do programa, altere a linha 1060 para chamar a sub-rotina 8000 em vez de 6000.

```
8000 C=-5
8010 FOR I=1 TO M(P)
8020 K=VAL(MID$(M$(P),I,1))
8030 IF K=Q THEN GOTO 8060
8040 GOSUB 9000
8050 IF E>C THEN J=K:C=E
8060 NFXT I
8070 RETURN

9000 E=ABS(P-B(K)-INT((P-1)/3)*3+INT((B(K)-1)/3)*3)
9010 E=E+ABS(INT((P-1)/3)-INT((B(K)-1)/3))
9020 F=ABS(K-B(K)-INT((K-1)/3)*3+INT((B(K)-1)/3)*3)
9030 F=F+ABS(INT((K-1)/3)-INT((B(K)-1)/3))
9040 E=F-E
9050 RETURN
```

A sub-rotina 8000 examina cada movimento possível, e chama a sub-rotina para calcular a mudança que seria produzida na distância se o movimento fosse feito. O movimento com a maior mudança é escolhido pela linha 8050. Os detalhes do cálculo na sub-rotina 9000 podem parecer difíceis, mas, tudo que está acontecendo é que a distância em quarteirões é calculada uma vez para cada posição (linhas 9000/9010 e 9030/

9040) e então é tomada a diferença (linha 9040). A tabela 2.1 mostra a estrutura de sub-rotina do programa completo.

TABELA 2.1

Sub-rotina	*Descrição*
1000	Estabelece o tabuleiro e embaralha as posições dos quadrados
2000	Imprime o tabuleiro
3000	Estabelece a tabela de movimentos legais M$ e a tabela de números dos movimentos M
4000	Encontra a posição atual do 'espaço' e a coloca em P
5000	Verifica a posição vencedora, S = O indica que a posição final foi alcançada
7000	Executa o movimento do quadrado que está em M(J) para M(P)
8000	Encontra o 'melhor' movimento
9000	Calcula a função de avaliação

```
8 2 3          . . .
9 5 7          . . .
6 1 4          . . .
               . . .
9 2 3          9 1 2
8 5 7          4 5 3
6 1 4          7 8 6

2 9 3          1 9 2
8 5 7          4 5 3
6 1 4          7 8 6

2 3 9          1 2 9
8 5 7          4 5 3
6 1 4          7 8 6

2 3 7          1 2 3
8 5 9          4 5 9
6 1 4          7 8 6

2 3 7          1 2 3
. . .          4 5 6
. . .          7 8 9
. . .
. . .          SOLUC. EM 51
```


Se você executar esta versão do programa, pode ter sorte e ver a solução antes de 100 movimentos serem executados. Por outro lado, pode não conseguir uma solução, mesmo depois de milhares de movimentos; isto é uma heurística, não um algoritmo, e sendo assim não pode garantir uma solução. Em média, o programa resolverá um, entre cada três programas, em menos de 100 movimentos (veja a figura 2.1), então, se não tiver sorte, tente novamente com um tabuleiro diferente.

Avaliação

Mudar do uso de movimentos aleatórios para a aplicação de uma heurística envolve muito pouca programação extra, mas realmente fornece um dramático aperfeiçoamento no desempenho. O exemplo que a seleção de movimentos aleatórios não pôde resolver em mais de cem mil movimentos é solucionado em 51 movimentos usando a heurística.

Isto não quer dizer que não haja problemas com o método. Algumas destas dificuldades são, elas próprias, instrutivas, e merecem um estudo posterior. Por exemplo, algumas vezes o computador fica repetindo o mesmo conjunto de movimentos muitas e muitas vezes. Uma razão para isto é que, se houver um empate na busca do "melhor" movimento, o movimento que vem primeiro na tabela de movimentos legais é sempre tomado. Isto poderia ser mudado, por exemplo, para uma seleção aleatória de movimentos empatados. Lembre-se de que não há garantias de que a heurística solucionará o problema, mas ela é melhor que a seleção aleatória, e, certamente, muito melhor que não ter solução alguma.

A heurística usada para solucionar o problema dos quadradinhos mostra muitas das características de heurísticas aplicadas a outros problemas. Sua simplicidade é uma vantagem por torná-la fácil de aplicar, mas vale a pena tentar aproveitá-la mais. Apesar da maneira pela qual você pode aprimorar a aplicação de uma heurística muitas vezes depender da natureza exata do problema, existe um método muito geral que é apropriado sempre que uma heurística está sendo aplicada em uma

seqüência de passos, cada um tentando chegar mais perto da solução. Este método se baseia na idéia de examinar as conseqüências do passo atual em passos futuros.

Pensando à frente

A heurística que usamos para solucionar o jogo dos quadradinhos era muito simples: examinar cada movimento possível, e tomar aquele que produz a máxima redução na distância entre o arranjo atual e o "objetivo" (arranjo final). Isto é intuitivamente simples, porque cada movimento tenta nos levar mais perto da solução. No entanto, se você observar uma pessoa jogando o jogo dos quadradinhos, verá que, freqüentemente, ela faz movimentos que aumentam a distância. O que isto sugere é que, às vezes, vale a pena afastar-se da solução temporariamente, se isto trouxer vantagens para movimentos futuros. Às vezes é melhor escolher um movimento que o afaste temporariamente da solução, mas que conduza você a um movimento realmente bom mais tarde. A conclusão que se tira destas observações é que não é sempre suficiente avaliar as vantagens do próximo movimento. Você tem que avaliar um movimento em termos dos movimentos que podem segui-lo. Em outras palavras, você tem que pensar no futuro.

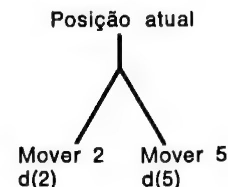
Isto parece muito difícil, mas, de fato, é simplesmente uma repetição de aplicações de métodos que nós já usamos antes.

A Árvore de Movimentos

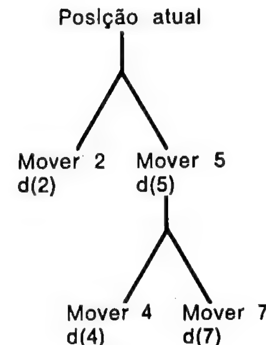
Imagine que o jogo tenha alcançado a seguinte posição:

6	3	1
2	4	8
	5	7

Existem dois movimentos possíveis a considerar. Você poderia mover ou o "2" ou o "5". Podemos considerá-los como dois diferentes "caminhos" que poderíamos tomar "rumo" à uma solução.



Cada movimento resulta em sua própria distância (possivelmente a mesma) da posição final, e é isto que usamos para escolher qual rumo tomaremos. (A distância após mover o quadrado n é escrita como d(n) no diagrama.) Após escolher algum dos movimentos, confrontamo-nos novamente com uma decisão entre vários outros movimentos. Por exemplo, se tomarmos o movimento "5", os novos movimentos possíveis serão "4" e "7". O movimento "5" deve ser rejeitado, porque não queremos voltar exatamente para o lugar de onde viemos! Podemos adicionar estas duas escolhas ao diagrama:



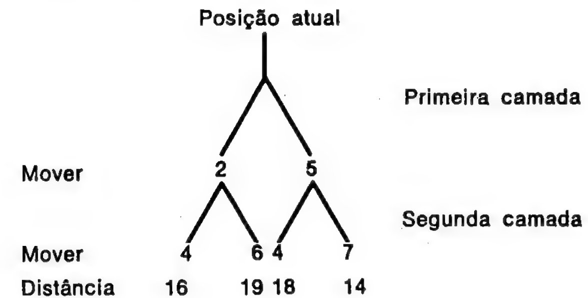
O diagrama dos movimentos possíveis é conhecido por "árvore" de movimentos", porque ele se parece com uma árvore desenhada de cabeça para baixo. Cada movimento forma um ponto de decisão, ou "nó", para o próximo estágio do jogo. Se você fosse desenhando nós em uma árvore de movimentos, eventualmente alcançaria o arranjo desejado dos quadrados, isto é, a distância seria zero. O problema do jogo dos quadradinhos pode ser visto

agora sob uma nova luz. O que estamos fazendo é pesquisando a árvore de movimentos, para encontrar uma rota (pode haver mais de uma) para o nó final, ou "terminal", que nos dá o arranjo final. O problema de examinar uma árvore para encontrar uma solução é um problema que surge muitas vezes de repente em AI, como resultado de situações que são superficialmente muito diferentes.

Aprofundando o exame

A melhor maneira de pensar sobre a escolha de movimentos com uma certa quantidade de "visão futura" é através de uma árvore de movimentos. Ao invés de avaliar cada um dos possíveis próximos movimentos, transferimos nossa atenção para as vantagens que os movimentos subseqüentes possam ter. Isto significa que, para cada movimento que podemos escolher, deveremos avaliar as possíveis vantagens de todos os movimentos aos quais eles conduzem. Isto pode parecer fácil, mas a dificuldade consiste em que o número de movimentos a considerar torna-se rapidamente enorme. Por exemplo, se existirem duas escolhas possíveis em cada nó, olhando um nó à frente haverá dois movimentos a considerar, olhando dois nós à frente já haverá quatro, três nós darão oito, e assim por diante. Isto não é tão ruim no caso do jogo dos quadradinhos, mas, em um jogo como xadrez, o número de movimentos possíveis em cada nó será da ordem de 10 ou 20, o que significa que haverá pelo menos 1000 diferentes movimentos a considerar, olhando somente três nós à frente! Cada vez que examinarmos um movimento à frente, estaremos considerando a próxima "camada" da árvore. Examinar uma camada à frente é freqüentemente chamado de uma busca de "uma camada" e, de modo semelhante, examinar "n' movimentos à frente é denominado de busca de "n camadas". Mais adiante neste capítulo, o programa dos quadradinhos será estendido de sua forma atual de busca de uma camada para uma busca de duas camadas, de modo que este novo tipo de busca merece uma consideração mais detalhada.

Voltando ao arranjo dos quadradinhos usado anteriormente, a árvore de duas camadas para a escolha do próximo movimento será:



Se adotarmos o movimento "2", o próximo movimento poderá ser "4" ou "6". Escolhendo o movimento "4", a distância será reduzida a 16, que é menor do que 19, que resultará da escolha do movimento "6" aí. Assim, se adotarmos o movimento "2", o melhor que poderá ser conseguido no próximo movimento é reduzir a distância a 16. Se, entretanto, adotarmos os movimentos "5" na primeira camada, o próximo movimento poderá ser "4" ou "7". O movimento "4" reduz a distância a somente 18, mas a escolha do movimento "7" irá reduzi-la a 14. Desta maneira, a escolha do movimento "2" não permitiria nada melhor do que reduzir a distância a 16 no próximo movimento, mas a escolha do movimento "5" poderia reduzir a distância a 14 no mesmo estágio. Assim, deve ser adotado o movimento "5"!

As distâncias associadas com cada movimento examinado, ou seja, 16 com "2" e 14 com "5", não são as distâncias obtidas imediatamente após estes movimentos, mas sim as distâncias obtidas após os próximos melhores movimentos terem sido executados. Como estas distâncias vêm, na realidade, de níveis inferiores da árvore, são freqüentemente chamadas de "distâncias refletidas", ou de "escores refletidos".

Após executado o movimento do quadradinho "5", o procedimento todo deve ser recomeçado, examinando-se dois movimentos adiante. Você pode estar pensando que, se já sabemos quais os melhores movimentos para os dois movimentos seguintes, o melhor será executá-los ambos, ou seja, mover "5" e,

em seguida, "7". A resposta a esta questão é que, após movido o quadradinho "5", podemos julgar a conveniência do movimento "7" à luz da distância refletida de um movimento adiante na árvore, e isto poderia provar que, perante esta consideração, o movimento "4" poderia ser o melhor no caso! Obviamente há uma certa contradição em escolher um movimento baseando-se no fato de o próximo movimento dar o melhor resultado e, depois, não executar este próximo movimento; e as implicações deste fato serão discutidas no próximo capítulo.

Um Programa para o Jogo dos Quadradinhos Usando Busca de 2 Camadas

A modificação do programa usando busca de uma camada para examinar a árvore de movimentos de duas em duas camadas não é muito difícil. O que deve ser feito é examinar cada movimento por sua vez a calcular a distância refletida. A maneira mais fácil de fazer isto consiste em mover temporariamente o quadradinho em questão e então usar as sub-rotinas originais para examinar os movimentos subseqüentes possíveis. Após encontrado o movimento subseqüente que minimiza a distância, bastará reverter o movimento temporário para restaurar a situação anterior do tabuleiro, antes de testar o próximo movimento. Após todos os movimentos possíveis terem sido tratados desta maneira, será adotado o que resultar na menor distância refletida. A única complicação que aparece é que o programa anterior economiza tempo não calculando a distância total do arranjo alvo, mas somente a variação de distância resultante de qualquer movimento (sub-rotina 9000). Assim, ao invés de tentar minimizar a distância refletida, estaremos, de fato, tentando maximizar a "variação de distância" refletida.

Resumindo, a versão final do programa deveria

(1) examinar cada movimento possível por vez, executando-o temporariamente e, então, analisando a variação de distância produzida por cada um dos movimentos subseqüentes possíveis.
(2) somar a variação máxima em distância obtida pelo movimento subseqüente à variação de distância obtida pelo movimento inicial, resultando a variação de distância refletida.

(3) finalmente, após examinados todos os movimentos iniciais possíveis, escolher aquele que produzir a variação máxima de distância refletida.

Isto pode parecer muito complicado, mas se você pensar um pouco sobre o assunto, ele acabará fazendo algum sentido. O que é surpreendente é que é possível implementar esta nova heurística com algumas poucas modificações ao programa anterior. Uma lista revista de sub-rotinas pode ser vista na Tabela 2.2.

A maior dificuldade vem da necessidade de executar temporariamente um movimento e avaliá-lo, sem que o resto do programa "pense" que este é o movimento real. A melhor maneira de evitar este problema consiste na introdução de algumas variáveis novas para armazenar o estado "real" do tabuleiro, ou seja, usar P1 para registrar a posição do quadrado (representado pelo 9) e P para registrar sua posição temporária.

As alterações ao programa original podem ser vistas abaixo:

```
65 Q1=0

120 IF S=0 THEN GOTO 190
130 PRINT
140 GOSUB 8500
150 K=J1:J=J1:P=P1:Q1=P1
160 GOSUB 9000
170 GOSUB 7000
180 P1=P:M=M+1:GOSUB 2000
185 GOTO 110

4010 IF B(I)=9 THEN P1=I

8500 C1=-5
8510 FOR Z=1 TO M(P)
8520 J0=VAL(MID$(M$(P1),Z,1))
8530 IF Q1=J0 THEN GOTO 8630
8540 J=J0:P=P1:K=J
8550 GOSUB 9000
8560 CT=E
8570 GOSUB 7000
8580 GOSUB 8000
8590 CT=CT+C
8600 IF CT>C1 THEN J1=J0:C1=CT
8610 J=P1:P=J0
8620 GOSUB 7000
8630 NEXT Z
8640 RETURN
```

Aparece somente uma nova sub-rotina (8500), mas o programa principal também foi modificado, seguindo as mesmas linhas do anterior, mas chamando a sub-rotina 8500 no lugar da 8000, para avaliar cada um dos movimentos possíveis. Usa ainda Pl para armazenar a posição do quadradinho e Jl como o movimento atual. A sub-rotina 8500 "varre" todos os movimentos possíveis usando um ciclo FOR. Inicialmente, cada movimento é avaliado usando a sub-rotina 9000 e o resultado armazenado em "CT" e, em seguida, o movimento é efetivamente executado pela sub-rotina 7000. Aí a sub-rotina 8000 é chamada para avaliar todos os movimentos subseqüentes, da mesma maneira usada para avaliar movimentos simples no programa original. Na saída da sub-rotina 8000, "C" conterá a variação máxima de distância e, embora não seja realmente usado, "J" conterá o movimento que produziria esta variação de distância. Em seguida, a variação máxima de distância produzida pelo movimento subseqüente é somada à variação de distância produzida pelo movimento original, guardado em CT e é feito um teste para ver se esta variação total é maior que a maior anteriormente encontrada (armazenada em C1). Se isto acontecer, a nova variação tornar-se-á a maior até este instante e o número do movimento inicial correspondente, que está em JO, será registrado em J1. Aí o movimento temporário executado no início do ciclo FOR será desfeito, usando-se para isto novamente a sub-rotina 7000, e então, será avaliado o próximo movimento possível. Ao final da sub-rotina 8500, o melhor movimento possível estará registrado em Jl e a melhor variação correspondente após um movimento subseqüente em Cl.

Avaliação e Sugestões

Se você experimentar esta versão de duas camadas do jogo dos quadradinhos, verificará que ela parece executar, na média, movimentos "melhores", ou mesmo, "mais inteligentes" que os da versão anterior. Entretanto, terá ainda uma tendência desagradável de "encalhar" em algum canto, repetindo a mesma seqüência de movimentos indefinidamente.

TABELA 2.2

Sub-rotina	*Descrição*
1000	Estabelece um novo tabuleiro e embaralha as posições dos quadradinhos, estando o número máximo de movimentos para resolver o problema em M1
2000	Imprime o tabuleiro
3000	Cria a tabela de movimentos legais
4000	Descobre a posição atual do "espaço" e armazena-a em P
5000	Testa se a posição final foi alcançada, S=O indica que foi
6000	Seleciona aleatoriamente um movimento P, J
7000	Executa o movimento do quadradinho, de B(J) para B(P)
8000	Descobre a "melhor" variação de distância produzida por um movimento de "segundo nível"
8500	Descobre a "melhor" variação refletida de distância para cada movimento possível
9000	Calcula a função de avaliação para o movimento do quadradinho, de B(K) para B(P) e armazena o resultado em E

(Isto acontece em aproximadamente metade dos casos, se os quadradinhos forem embaralhados mais do que 50 movimentos ao acaso.) Há três possíveis soluções para este problema:

(1) procurar detetar o estado "encalhado" e, então, executar um movimento aleatório

(2) procurar melhorar a função de avaliação para evitar encalhar

(3) aumentar a profundidade da busca para três, ou até quatro, camadas.

Entretanto, o jogo dos quadradinhos já alcançou o limite de sua utilidade como um exemplo simples e, assim, quaisquer aperfeiçoamentos adicionais são deixados como exercício, caso você esteja interessado em procurar executá-los.

Heurísticas em geral

Neste exemplo prático podem ser vistas muitas características e dificuldades das soluções heurísticas. Em geral, é necessário encontrar uma regra cuja aplicação repetida tenda a levar o processo para mais perto da solução, ou alvo. Embora isto

pareça fácil, muitas vezes é difícil compreender o problema com suficiente clareza para encontrar e aplicar uma heurística. Por exemplo, nem sempre é possível medir a distância da situação atual para a solução. Quase sempre é possível concluir que uma solução foi alcançada, mas saber se um movimento conduzirá para mais perto ou mais longe de uma solução é algo que freqüentemente é difícil de conseguir. Na verdade, às vezes pode ser difícil até entender o que constitui um "movimento"! No jogo dos quadradinhos e em muitos outros exemplos incluídos neste livro, a gama de movimentos válidos é definida pelas regras do jogo. Estas regras não somente indicam quais movimentos são proibidos, mas dão uma lista de todos os movimentos considerados legais, da qual pode ser selecionado aquele que, espera-se, seja melhor que os outros. O problema com a vida real é que as regras raramente são simples, podendo não ser possível enumerar todos os movimentos possíveis e muito menos compará-los para achar o melhor!

Mesmo que seja difícil aplicar o método heurístico em alguns casos, pelo menos ele nos dá uma alternativa para quando o método algorítmico, que nos é mais familiar, falha. Resumindo, a heurística é uma ferramenta imprescindível na solução dos problemas humanos e, se os computadores forem realmente transformar-se em máquinas pensantes, deverão, inevitavelmente, empregar os métodos imprecisos da heurística, lado a lado com os algoritmos exatos.

3

Quando as heurísticas se encontram: a estratégia de competição

O capítulo anterior introduziu a idéia do uso de uma heurística. Esta maneira de atacar problemas é bastante geral e permite a solução de uma gama destes. No entanto, não sabemos ainda como tratar problemas que envolvem a interação entre dois atores. Em termos mais familiares, não sabemos como escrever programas para jogos competitivos. Por exemplo, o jogo dos quadradinhos, descrito no capítulo anterior, pode ser facilmente resolvido por uma heurística simples que, a cada passo, leva para mais perto da solução. No caso de um jogo tal como xadrez, ou damas, entretanto, este enfoque simples falharia. A razão para isto é que você pode usar uma heurística que o leve para mais perto da *sua* solução (ou seja, ganhar o jogo), mas seu adversário procurará desfazer sua vantagem e, se possível, levar o jogo para mais perto da solução *dele*. É possível jogar xadrez usando uma heurística e ignorando completamente os movimentos do adversário, mas é improvável que isto nos leve à vitória! É claro que, para estender o método heurístico aos jogos competitivos — dos quais os mais simples são os que envolvem somente duas pessoas — é necessário encontrar uma maneira de não somente considerar a vantagem obtida com o próximo movimento, mas também como o próximo movimento do adversário poderá afetá-la.

Um Jogo Simples para duas Pessoas: o Jogo da Velha

É difícil encontrar problemas que sirvam como exemplos adequados. A maioria dos problemas, ou são excessivamente difí-

ceis de se entender rapidamente, ou muito fáceis, não apresentando nenhum desafio importante para o método. O caso dos jogos de duas pessoas não constituem exceçao a esta regra. Xadrez é claramente muito difícil e não pode ser atacado com nada menos do que linguagem de máquina. Damas é um pouco mais fácil, mas resulta ainda em um programa muito longo. Um jogo que apresenta o nível conveniente de dificuldade é o conhecido "jogo da velha". Este é o jogo mais difícil para duas pessoas que pode ser programado em BASIC em um número suficientemente pequeno de linhas para que seja fácil de entender. Tem também uma variedade suficiente de possibilidades para permitir a ilustração de algumas técnicas mais avançadas de AI. Entretanto, apresenta também o problema de existir um algoritmo suficientemente pequeno que, no pior dos casos, resulta num empate e, se o adversário cometer um erro, na vitória. Para fins de exemplificação, entretanto, a solução algorítmica será ignorada. Se, ao final do capítulo, você sentir a necessidade de um exemplo mais complexo, pode tentar estender os programas para o jogo da velha tridimensional — um real desafio, principalmente para imprimir o tabuleiro.

O Jogo da Velha

Antes de continuar a ler este capítulo, seria interessante você convencer alguém a jogar algumas partidas do jogo da velha com você, para refrescar sua memória sobre as considerações que regem o jogo. Se isto não for possível, não se preocupe muito, porque não é difícil entender os pontos importantes do jogo. Quem está jogando "O" procura colocar três "O"s alinhados, segundo uma fila, uma coluna, ou mesmo uma diagonal. Obviamente, no caminho para uma posição vencedora, qualquer linha que possa ser convertida em uma linha vencedora é desejável. Por exemplo, dois "O"s e um branco, ou um "O" e dois brancos em uma linha são boas posições. Entretanto, como este é um jogo de duas pessoas, você não pode pensar somente em colocar os "O"s nas melhores posições para completar sua linha, mas terá também que jogar para bloquear a linha que seu adversário estiver tentando completar. Enquanto isto, ele estará fazendo o mesmo.

Uma Heurística para o Jogo da Velha

Para programar jogos de duas pessoas são necessárias heurísticas na forma de selecionadores de movimentos que usem uma função de avaliação. Após a discussão anterior, é fácil concluir que qualquer função de avaliação deverá ser baseada no número de linhas com um ou dois "O's que uma jogada pode produzir. De fato, contando

o1 = o número de linhas com 1 "O"
o2 = o número de linhas com 2 "O"s

e

o3 = o número de linhas com 3 "O"s

que um movimento produz é fácil decidir sobre sua conveniência. Mas, e as linhas que o adversário está formando com "X"? Enfim, não teria sentido executar um movimento que resultasse em uma linha de 2 "O"s, se este movimento também resultasse em uma linha de 2 "X"s. (A razão para tal é clara: no próximo movimento, que será do adversário, a linha de 2 "X"s será convertida em uma linha de 3 "X"s e ele terá ganho o jogo!) Assim, fica clara a necessidade de considerar também alguma informação sobre a situação atual dos "X"s, para avaliar a conveniência ou não de um movimento. Da mesma maneira que a posição dos "O"s pode ser resumida contando-se as linhas de "O"s, a posição dos "X"s será resumida contando-se as linhas de "X"s:

x1 = o número de linhas com 1 "X"
x2 = o número de linhas com 2 "X"s

e

x3 = o número de linhas com 3 "X"s.

Para se chegar a uma função de avaliação, é necessário encontrar alguma maneira de combinar as medidas individuais em um único número que cresce à medida que a posição dos "O"s melhora e diminui quando ela piora. O método mais simples consiste em somar todas as medidas, precedidas de sinais + ou — apropriados. Por exemplo, poderia ser usada

$$ev = o3 - x2 + o2\ x1 + o1$$

que combina o3, o2 e o1 como medidas positivas da situação dos "O"s, e x2 e x1 como medidas negativas de sua situação. (Observe que não há necessidade de incluir x3 na soma, pois se houver uma linha com 3 "X"s o jogo já estará perdido para os "Os"s.) A desvantagem desta avaliação é fácil de ver: ela ignora a importância relativa das diversas medidas. Por exemplo, uma única linha com 3 "O"s é melhor do que qualquer outra coisa que possa acontecer, pois o movimento que a puder produzir levará à vitória no jogo! Felizmente, é fácil também introduzir a importância relativa das medidas na função, multiplicando-se cada parcela por uma constante, ou seja, transformando-a em

$$ev = a*o3 - b*x2 + c*o2 - d*x1 + e*o1$$

sendo o único problema remanescente a escolha dos valores de a, b, c, d e e. É muito comum o uso de funções de avaliação na forma de uma soma de parcelas referentes a diferentes medidas, cada uma recebendo um peso de acordo com sua importância. No caso atual, e de acordo com a discussão anterior, uma linha de 3 "O"s deve receber um peso tão grande que reflita o fato de este ser o resultado mais desejado de um movimento. Seu peso deve ser suficiente para superar o efeito de quaisquer outras medidas, mesmo que o3 tenha somente o valor de 1. Para simplificar as coisas, adotemos o valor de 128 para a. É mais difícil chegar a valores adequados para b, c, d e e. Obviamente, a próxima coisa mais importante que pode acontecer é x2, pois um movimento que resulte em (ou melhor, deixe) uma linha de 2 "X" é realmente *muito infeliz*. Assim, o valor de "b" deve ser grande, mas não tão grande que possa interferir com o efeito de a*o3. Se existir uma linha com 3 "O"s, não é difícil ver que poderão existir no máximo duas linhas com 2 "X"s, de maneira que, desde que $2*b < a$, a presença da linha com 3 "O"s superará as presenças dos 2 "X"s. (Se existir uma linha de 3 "O"s e duas linhas de 2 "X"s, não poderá haver linhas com 1 de nenhuma dos dois para atrapalhar os cálculos!) Desta forma, um valor conveniente para "b" é 63. Por raciocínio semelhante, chega-se aos valores $c=31$, $d=15$ e $e=7$, resultando a função de avaliação definitiva:

$$ev = 128*o3 - 63*x2 + 31*o2 - 15*x1 + 7*o1$$

Esta função de avaliação pode ser aplicada ao jogo da velha da mesma maneira que a função de avaliação do jogo dos quadradinhos foi aplicada ao seu programa de uma camada no Capítulo 2. Entretanto, antes de proceder para considerar um programa de uma camada para este novo jogo em detalhe, é importante observar que a atribuição de valores a pesos tais como os de *a* e *e* freqüentemente exige um processo de tentativa e erro. Se o uso de uma função de avaliação resultar na perda de muitos jogos, isto será sinal de que os pesos deverão ser reajustados.

Um Jogo da Velha de uma Camada

Agora que já temos uma função de avaliação é possível escrever um programa de uma camada para o jogo da velha. Excetuando-se somente as diferenças nos detalhes, este segue a mesma linha do programa do jogo dos quadradinhos de uma camada. Inicialmente "X" (o humano) faz um movimento e, então, cada um dos possíveis movimentos que "O" pode fazer em respostas será avaliado. O movimento selecionado será o que resultar no maior valor da função de avaliação, ou seja, "O" seleciona o movimento que maximizar a função de avaliação.

O programa resultante não é tão longo como você poderia esperar.

```
  10 DIM X(4),Y(4)
  20 GOSUB 8000
  30 GOSUB 6000
  40 GOSUB 7000
  50 GOSUB 5000
  60 GOSUB 7000
  70 GOTO 30

4000 FOR K=1 TO 4:X(K)=0:Y(K)=0:NEXT K
4010 FOR L=1 TO 3
4020 S=0
4030 T=0
4040 FOR  K=1 TO 3
4050 IF A(L,K)=1 THEN S=S+1
4060 IF B(L,K)=1 THEN T=T+1
4070 NEXT K
4080 IF S=0 THEN Y(T+1)=Y(T+1)+1
4090 IF T=0 THEN X(S+1)=X(S+1)+1
4100 NEXT L
```

```
4110 FOR L=1 TO 3
4120 T=0
4130 S=0
4135 FOR K=1 TO 3
4140 IF A(K,L)=1 THEN S=S+1
4150 IF B(K,L)=1 THEN T=T+1
4160 NEXT K
4170 IF S=0 THEN Y(T+1)=Y(T+1)+1
4180 IF T=0 THEN X(S+1)=X(S+1)+1
4190 NEXT L
4200 GOSUB 4300
4210 GOSUB 4400
4220 E=128*Y(4)-63*X(3)+31*Y(3)-15*X(2)+7*Y(2)
4230 RETURN

4300 T=0
4310 S=0
4320 FOR K=1 TO 3
4330 T=T+A(K,K)
4340 S=S+B(K,K)
4350 NEXT K
4360 IF S=0 THEN X(T+1)=X(T+1)+1
4370 IF T=0 THEN Y(S+1)=Y(S+1)+1
4380 RETURN

4400 T=0
4410 S=0
4420 FOR K=1 TO 3
4430 T=T+A(4-K,K)
4440 S=S+B(4-K,K)
4450 NEXT K
4460 IF S=0 THEN X(T+1)=X(T+1)+1
4470 IF T=0 THEN Y(S+1)=Y(S+1)+1
4480 RETURN

5000 M=-256
5010 FOR J=1 TO 3
5020 FOR I=1 TO 3
5030 IF A(I,J)=1 OR B(I,J)=1 THEN PRINT "OC":
     GOTO 5090
5040 B(I,J)=1
5050 GOSUB 4000
5060 PRINT E
5070 IF E>M THEN M=E:A=I:B=J
5080 B(I,J)=0
5090 NEXT I
5100 NEXT J
5110 B(A,B)=1
5120 RETURN
```



```
6000 PRINT "SEU MOV. ";
6010 INPUT I,J
6020 A(I,J)=1
6030 RETURN

7000 FOR J=1 TO 3
7010 FOR I=1 TO 3
7020 IF A(I,J)=1 THEN PRINT "X";
7030 IF B(I,J)=1 THEN PRINT "O";
7040 IF A(I,J)+B(I,J)=0 THEN PRINT " ";
7045 PRINT ".";
7050 NEXT I
7060 PRINT
7070 NEXT J
7080 RETURN

8000 DIM A(3,3),B(3,3)
8010 GOSUB 7000
8020 RETURN
```

Mais uma vez, o programa é escrito como uma coleção de sub-rotinas, de maneira que fique fácil utilizá-lo como "banco de testes" para quaisquer outras idéias que se deseje tentar. A Tabela 3.1 mostra sua estrutura geral.

O programa principal (10 — 70) é simplesmente uma lista de chamadas de sub-rotinas. A sub-rotina 8000 é uma sub-rotina geral de estabelecimento do programa. A sub-rotina 6000 recebe e executa os movimentos de "X". A sub-rotina 7000 imprime o tabuleiro. A função de avaliação é calculada pela sub-rotina 4000, que é chamada pela sub-rotina 5000 para cada movimento possível que "O" pode fazer em resposta ao movi-

TABELA 3.1

Sub-rotina	*Descrição*
3000	Busca de duas camadas da árvore dos movimentos
4000	Calcula a função de avaliação
5000	Busca de uma camada da árvore dos movimentos
6000	Recebe os movimentos de "X"
7000	Imprime o tabuleiro
8000	Estabelece o jogo

mento de "X". Os detalhes do programa são compreendidos facilmente desde que se saiba que o tabuleiro é representado, de fato, por dois *arrays* diferentes, "A" e "B". "A" é usado para registrar a posição de "X" e "B" para a de "O". O uso de dois *arrays* permite que ambos "X" e "O" sejam representados por "1", o que simplifica alguns cálculos. A sub-rotina 4000 calcula a função de avaliação para cada situação atual do tabuleiro, de maneira que, para avaliar um movimento possível, é necessário executá-lo antes de chamar a sub-rotina 4000 e, posteriormente, lembrar de desfazê-lo. Embora haja várias maneiras de acelerar o cálculo da função de avaliação, a sub-rotina 4000 foi escrita para ser clara e não para ser rápida, e trabalha contando quantos de cada símbolo existem em cada fila, em cada coluna e em cada diagonal. O *array* "x" é usado para contar, em x(1), quantas linhas há sem nenhum "X", em x(2) quantas linhas há com somente um "X" e assim por diante até x(4), que registra quantas linhas há com 3 "X"s. Da mesma maneira, o *array* "y" conta o número de linhas de "O" presentes no tabuleiro. Observe que, para que uma linha seja contada como tendo qualquer número de "X"s, ela não poderá ter nenhum "O" e vice-versa. O programa imprime o valor da função de avaliação para cada movimento considerado, para fins de perfeita compreensão do processo e imprime "oc" quando encontra uma posição ocupada, não constituindo, pois, um movimento possível.

O programa não recebeu "acabamento final", no sentido de que não há nenhum teste de validade dos movimentos, nem de término do jogo, de maneira que há muito espaço para aperfeiçoamento. Entretanto, mesmo na sua presente simples forma é suficiente para investigar a conveniência da função de avaliação.

Avaliação

Após jogar algumas partidas contra este simples programa, você ficará surpreso com sua eficiência! Esta é mais o resultado da simplicidade do jogo da velha, do que da capacidade do método usado no programa. Entretanto, haverá, pelo menos um jogo que o programa perde, e é através do estudo dos exemplos onde ele falha que seu aprimoramento pode ser feito. Se

"X" jogar 1,1 o programa responderá com 2,2. Se, então, "X" executar a estranha jogada 3,3 o programa responderá com 3,1, resultando na posição

X		O
	O	
		X

e o jogo estará perdido, pois, então, "X" jogará 1,3, "O" responderá com 1,2 e "X" ganhará jogando 2,3.

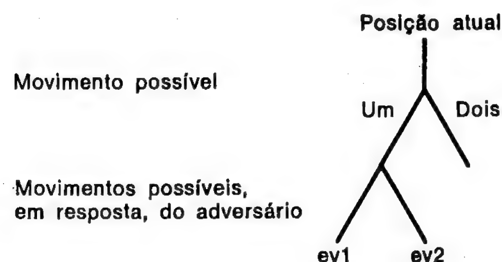
Uma Solução com Duas Camadas: Minimax

Acompanhando os argumentos apresentados no Capítulo 2, uma maneira de aperfeiçoar a heurística incorporada na função de avaliação consiste em aplicá-la a uma busca de duas camadas na árvore dos movimentos. No caso presente, o único problema é que o próximo movimento é escolhido pelo adversário, estando fora de nosso controle. Poder-se-ia concluir, neste ponto, que não há nada a fazer porque, embora possamos avaliar nosso próximo movimento, é impossível considerar também o próximo movimento do adversário, pois não é possível determiná-lo. Entretanto, supondo que o adversário é razoavelmente competente no jogo, é claro que irá executar um movimento que seja "bom" para sua posição após nosso movimento. Como o que é bom para nosso adversário é ruim para nossa posição, esta é outra maneira de dizer que, em seguida a qualquer movimento que façamos para melhorar nossa posição, nosso adversário fará um movimento que procure tornar nossa posição a pior possível.

Isto significa que, após encontrarmos um próximo movimento que resulte no maior valor possível da função de avaliação, podemos ficar desapontados descobrindo que o próximo movimento de nosso adversário reduzirá consideravelmente seu valor. Obviamente, perante esta situação, terá mais sentido não executar a jogada que maximizar a função de avaliação, mas aquela que resulte no máximo valor da função de avaliação após nosso adversário ter executado *seu* movimento mais devastador. Sendo este o que *minimiza* nossa função de avaliação, estaremos, na realidade, executando a jogada que maximizará

a função de avaliação após nosso oponente ter tentado minimizá-la. Por razões óbvias, esta estratégia é conhecida por *minimax*.

A idéia de maximizar o mínimo não é fácil de compreender, de maneira que é útil examiná-la novamente, desta vez em termos da árvore de movimentos. Um programa de duas camadas deverá examinar, não somente cada movimento que poderá executar — ou seja, prosseguir um nível abaixo na árvore — mas também cada resposta que o adversário poderá executar, ou seja, dois níveis abaixo na árvore. No caso de um jogo não competitivo, o programa tentaria maximizar a função de avaliação na posição dois níveis adiante, mas, no caso de um jogo competitivo, o segundo movimento é o do adversário, que procurará executar aquele que minimize nossa função de avaliação. Esta situação pode ser representada pela seguinte parte da árvore de movimentos:



(A árvore toda para o jogo da velha é muito maior do que isto!) Na situação mostrada, para avaliar o "movimento possível um", o programa terá que avaliar cada uma das respostas possíveis que o adversário poderia executar, calculando as funções de avaliação resultantes, ou seja, ev1 e ev2 na figura. Admitindo que ev1 dê 5 e ev2 dê 10, fica claro que o adversário escolherá o movimento um em resposta (pois este é o que minimizará nossa função de avaliação). Assim, se o programa escolher o movimento um, o resultado será ev1 (ou seja, 5) no próximo movimento. Usando a mesma terminologia introduzida para trabalhar por mais níveis da árvore de movimento do jogo dos quadradinhos, ev1 é o valor "refletido" da função de avaliação e o programa deverá escolher o movimento que maximizar este valor refletido.

Um Programa BASIC de Duas Camadas

Para produzir um programa BASIC que implemente esta busca minimax de duas camadas, basta reescrever as sub-rotinas de busca e de cálculo da função de avaliação, cujas novas versões são apresentadas em seguida.

```
  50 GOSUB 3000

3000 G=-255
3010 FOR J=1 TO 3
3020 FOR I=1 TO 3
3030 IF A(I,J)=1 OR B(I,J)=1 THEN PRINT "OC":
     GOTO 3180
3040 B(I,J)=1
3050 D=255
3060 FOR M=1 TO 3
3070 FOR N=1 TO 3
3080 IF A(M,N)=1 OR B(M,N)=1 THEN GOTO 3130
3090 A(M,N)=1
3100 GOSUB 4000
3110 IF E<D THEN D=E
3120 A(M,N)=0
3130 NEXT N
3140 NEXT M
3150 PRINT D
3160 IF D>G THEN G=D:A=I:B=J
3170 B(I,J)=0
3180 NEXT I
3190 NEXT J
3200 B(A,B)=1
3210 RETURN

4220 E=256*Y(4)-128*X(4)-63*X(3)+31*Y(3)-
     15*X(2)+7*Y(2)
4230 F=128*X(4)-63*Y(3)+31*X(3)-15*Y(2)+7*X(2)
4240 RETURN
```

Após as discussões do programa de uma camada e das idéias envolvidas na busca minimax, o funcionamento deste programa ficará claro depois de algum estudo. A sub-rotina 3000 toma cada movimento possível por vez e faz sua avaliação executando cada possível resposta e chamando a sub-rotina 4000 para calcular a função de avaliação. O resultado refletido é encontrado armazenando-se o mínimo da sub-rotina 4000 em "d". Este é então comparado com o máximo valor atual do escore refletido, armazenado em "g". Desta maneira é possível descobrir qual é o movimento que tem o escore refletido má-

ximo. Observe que, como a sub-rotina 4000 continua calculando a função de avaliação com base na situação do tabuleiro no instante em que é chamada, é necessário executar cada movimento possível, chamar a sub-rotina 4000 para avaliá-lo e depois lembrar de desfazê-lo para devolver ao tabuleiro a sua situação original. O único outro ponto de importância é a inclusão de uma contagem do número de linhas com 3 "X"s. No programa de uma camada não poderia haver linhas com 3 "X"s a serem avaliadas pela função de avaliação, já que, no caso, o jogo já teria terminado! Entretanto, no programa de duas camadas, que analisa duas jogadas adiante, tal desastre pode ser previsto, de modo que a função de avaliação deve ser modificada para considerar a importância de *nunca* executar um movimento que resulte em uma linha de 3 "X"s no próximo movimento do adversário.

Avaliando o Programa de duas Camadas

A primeira coisa que deve impressioná-lo com respeito ao programa de duas camadas é o fato de que ele é muito lento. Isto é parcialmente resultante de que, para fins de maior clareza, não se procurou economizar tempo no programa BASIC. Entretanto, é também resultante do fato de que a quantidade de trabalho necessária a uma busca de duas camadas é também maior. Esta observação está na base da maioria dos problemas encontrados na programação de jogos. Mesmo uma função de avaliação melhorará quando usada por mais níveis na árvore de movimentos, mas a quantidade de trabalho envolvido cresce em proporções astronômicas! É fácil decidir se vale a pena aceitar o trabalho adicional envolvido em uma busca de dois níveis, jogando-se as partidas que o programa de uma camada perde. Neste caso, o programa de duas camadas encontrará maneiras de evitar a derrota certa, mas deixarei ao leitor descobrir por quê.

Para Além do Jogo da Velha

O jogo da velha serviu bem como um exemplo, mas sua utilidade terminou com o programa de duas camadas. Ele simplesmente não é suficientemente difícil para merecer uma avaliação por mais níveis que exigisse mais programação. (Entretanto,

será novamente usado no próximo capítulo como um exemplo do problema, ligeiramente diferente, de escrever um programa que *aprende* a jogar um jogo.) Para ver como se desenvolve o assunto dos jogos competitivos e seus respectivos programas, é necessário considerar damas ou xadrez. Um programa que jogue tais jogos, por mais simples que seja, tem que examinar a árvore de movimentos por quatro ou mais movimentos à frente. Entretanto, examinar toda a árvore de movimentos a estes níveis requereria muitas horas de computação (mesmo que o programa fosse escrito em linguagem de máquina), de maneira que a maior parte do esforço de programação, nestes casos, é gasto na procura de maneiras de ignorar movimentos que são claramente inconvenientes. Estas técnicas de desprezar partes da árvore de movimentos são freqüentemente chamadas de "poda" da árvore, mas infelizmente estão além do objetivo deste livro.

Outras áreas de programação de jogos competitivos que são de interesse incluem métodos especiais para jogar a abertura e o final dos jogos. No caso do jogo da velha, o primeiro movimento executado por "O" será sempre o canto superior esquerdo ou o centro do tabuleiro, dependendo de qual estiver livre. Esta observação pode ser usada para acelerar o programa escrevendo-se uma sub-rotina especial para o primeiro movimento. Este é um exemplo de uma sub-rotina para um "jogo de abertura". No jogo de xadrez é comûm manter alguns dos primeiros movimentos das aberturas "clássicas" armazenadas na memória, de maneira a levar o jogo a um ponto no qual a função de avaliação possa assumir o controle da escolha dos próximos movimentos. Da mesma forma há várias estratégias especiais para finalizar um jogo e estas também são freqüentemente armazenadas nos programas de jogos, para assumir o controle quando o jogo chegar a um ponto em que seja facilmente levado à vitória.

Outra área que pode ser estudada é o aperfeiçoamento da função de avaliação à medida que o programa joga. Este resultado com um "quê" de mágico pode ser alcançado observando-se que o escore refletido dá uma idéia da conveniência de um movimento após a realização de alguns outros movimentos. O valor da função calculado no próprio movimento também dá uma idéia de sua conveniência, mas sem a vantagem de ver o que acontece adiante na árvore. Obviamente, uma boa função

de avaliação deve dar aproximadamente o mesmo resultado quando aplicada um movimento à frente que quando aplicada n movimentos à frente, para se obter o valor refletido. (Se este for o caso, serão obtidos resultados quase tão bons sem a necessidade de pesquisar adiante.) Se este não for o caso, o programa poderá tentar melhorar a qualidade da avaliação, ajustando os valores das constantes de sua função, desta maneira "aprendendo" a jogar melhor.

Heurística Competitiva

É fácil esquecer a dificuldade inerente ao jogo de xadrez e, assim, minimizar a conquista envolvida no desenvolvimento de programas para jogar xadrez que possam desafiar jogadores humanos. No momento, ainda não existem programas para jogar xadrez que possam ganhar os melhores jogadores humanos, mas mesmo o tipo de programa que pode ser comprado para computadores pessoais joga bastante bem. Em outras palavras, mesmo estes humildes micros são capazes de resolver problemas a um nível de sofisticação suficiente para desafiar um adversário humano. Nesta curta introdução às idéias envolvidas com o uso de heurísticas contra um adversário, foram discutidas muitas das que são usadas na escrita de programas que chegam perto do comportamento humano em determinada gama de tarefas supostamente inteligentes.

4
Pensamento e raciocínio: sistemas especialistas

Até o momento, os dois problemas clássicos da AI considerados — o jogo dos quadradinhos e o jogo da velha — podem levar à conclusão de que a inteligência artificial parece ocupar-se com a escrita de programas para duplicar uma parte muito pequena do comportamento humano, pois, embora os jogos constituam um tópico muito geral, logo que um deles passa a receber atenção particular, muito rapidamente parecerá necessário desenvolver teorias e métodos especiais para seu caso. Ao contrário disto, a principal tarefa de AI é, na realidade, procurar duplicar o tipo mais geral de atividade mental — o pensamento.

Até um certo ponto, os programas que jogam xadrez e outros jogos constituem tentativas de executar tarefas que os humanos normalmente diriam que exigem "pensamento". O problema é que, logo que sua maneira de operar é conhecida, fica difícil acreditar que eles trabalhem da mesma maneira que um humano. Será que os grandes mestres do xadrez, na prática, analisam os problemas deste jogo até seis ou mais movimentos à frente? Não estarão os programas para jogar xadrez simplesmente usando a velocidade do computador para executar uma análise na base da "força bruta" do jogo? Da mesma maneira que os programas de xadrez não jogam da maneira que um humano faria, muitos programas de AI parecem alcançar aproximadamente os mesmos resultados que os humanos, usando, entretanto, métodos que parecem completamente diferentes. É

importante compreender que, sempre que você chegar a entender de que maneira um programa de AI funciona, pode ser levado a concluir que sua operação é essencialmente simples — ou seja, qualquer coisa que você entende parece simples.

Tudo isto parece levar à conclusão de que talvez os programas de AI que foram examinados até agora não sejam absolutamente inteligentes, mas somente "espertos". Isto é verdadeiro em parte, mas efetivamente ignora alguns pontos essenciais. O cérebro humano é um aparelho muito complicado, e é bem capaz de executar um grande número de operações simples simultaneamente, sendo, portanto, perfeitamente possível que a inteligência humana seja o resultado de um grande número de operações simples executadas simultânea e repetidamente. Para o estudo de inteligência artificial, isto pode significar que, mesmo que algum dia seja descoberto o funcionamento exato do cérebro humano, não será possível usar este conhecimento para escrever programas, pois os computadores não os poderiam executar em um tempo razoável. Em outras palavras, a escolha pode ser entre fazer exatamente como os humanos e não conseguir um programa prático, ou usar os talentos especiais dos computadores para resolver os problemas de maneiras não-humanas. Assim, no caso do jogo de xadrez, é bem possível que um jogador humano efetivamente examine muitas jogadas à frente, mas sem considerar todos os movimentos possíveis. Ao contrário, usam uma heurística para "podar" a árvore dos movimentos para um tamanho admissível. A maioria dos programas de computador para xadrez examinam muitos dos movimentos constantes da árvore, já que usam algoritmos simples de "poda" (ao invés de heurísticas) para descobrir um bom movimento a executar. Desta forma, muito freqüentemente, um computador executará um movimento que, para um humano, não parece fazer parte de nenhuma "estratégia" de jogo. Devido a isto, os bons jogadores humanos tendem a identificar o jogo do computador como "não-humano", mas isto pode ser uma diferença simplesmente em ênfase, e não em tipo.

Assim, será que os programas de xadrez "pensam" em algum sentido no jogo que estão jogando? A resposta a esta pergunta depende, até certo ponto, da cautela ao se usar a palavra "pensar". O programa de xadrez usa alguns dos métodos que os humanos provavelmente usam; avaliação dos movimentos, exame de algumas partes da árvore dos movimentos etc.... Entretanto, para tornar o programa de xadrez prático, ele usa as exclusivas vantagens do computador digital de resolver aritmética rapidamente a fim de pesquisar a árvore de movimentos completamente até um determinado nível. Na minha opinião, os programas de xadrez não pensam sobre o jogo de xadrez, mas, de certa forma, compartilham algumas características com os jogadores humanos.

A mesma coisa acontece com outros programas de AI: usam alguns métodos humanos, mas sempre combinados com outros métodos adaptados para computadores. Pode ser que algum dia sejam escritos programas baseando-se somente em métodos humanos, mas isto não parece muito provável no futuro próximo. Por enquanto, os programas que jogam xadrez ou resolvem problemas em geral continuarão sendo escritos usando um pouco de esperteza misturada com força bruta.

Isto não significa que não possam ser escritos programas que dêem a impressão de pensar, pois, da mesma forma que é possível escrever programas que joguem xadrez ao nível de mestre, é bem possível escrever programas em qualquer área específica que "pensem" e resolvam problemas tão bem quanto os especialistas da área.

Um Programa Geral para Resolver Problemas

Não é difícil escrever um programa que resolva problemas de um tipo específico. Por exemplo, é possível escrever um programa para provar teoremas matemáticos. De fato, pesquisadores em AI já escreveram programas que "raciocinam" em áreas de matemática tão bem que ocasionalmente já encontraram provas melhores para teoremas já conhecidos ou mesmo encontraram provas para teoremas sem prova conhecida. Neste sentido, pode-se concluir que os programas de AI já superam a inteligência humana! Entretanto, é claro que, se um problema não relacionado com matemática for submetido a um programa de provar teoremas, provavelmente não sairá uma resposta razoá-

vel! O programa trabalha dentro de um "mundo" muito estreito, do qual não pode sair para considerar coisas diferentes. Entretanto, nos primeiros tempos da AI foi demonstrado que um programa geral para resolver problemas poderia ser escrito. Bastaria dar-lhe uma descrição do "mundo" sobre o qual deveria "raciocinar", e ele poderia começar a resolver problemas lógicos sem nenhuma dificuldade.

Humanos Resolvendo Problemas

Os computadores raciocinam usando lógica exata e é por isso que são tão bons em provar teoremas matemáticos. Os humanos, por outro lado, raramente pensam logicamente neste sentido, a menos que sejam forçados a isto pela natureza dos problemas. Isto não quer dizer que os humanos pensam de modo ilógico! O que tende a acontecer é que um humano não analisará a situação nas suas menores minúcias, mas procurará utilizar sua experiência passada no assunto. Procurará comparar algum detalhe do problema com algo já visto no passado, ou seja, fará uso de um amplo conhecimento sobre a maneira de funcionar do mundo real, conhecimento este raramente incluído em programas que raciocinam logicamente.

Sendo este o caso, antes de construir programas para raciocinar como os humanos, é necessário inicialmente examinar como é feita a representação do conhecimento nos computadores. Admite-se que os computadores são muito bons para colecionar e armazenar vastas quantidades de informaçõs e isto é realmente verdadeiro, mas todo este armazenamento é feito de maneira muito simples. Uma coleção de fatos armazenada em um computador parece-se mais com a maneira pela qual uma enciclopédia "lembra-se" das coisas do que com a maneira pela qual os humanos se lembram. Por exemplo, uma coleção de fatos é inútil, a menos que se saiba quais são as "conseqüências" de um fato. Se você estiver tentando decidir como ficará o tempo, tanto você como um computador poderão saber que o céu está negro e cheio de nuvens, mas somente você poderá deduzir destes fatos que é provável que chova. Em outras palavras, você (humano) conhece as conseqüências (possíveis)

de um céu negro e cheio de nuvens, o que o computador não conhece!

Não é difícil imaginar maneiras de armazenar informação juntamente com suas conseqüências, representadas por uma coleção de regras. Por exemplo, o "conhecimento" do tempo poderia ser armazenado em um computador como

IF céu negro e nublado THEN alta possibilidade de chuva

Em geral, uma peça de conhecimento pode ser representada por uma lista de condições e outra de conseqüências. Por exemplo

IF nuvens negras, alta umidade, verão THEN tempestade

poderia ser uma declaração sobre o que sabemos acerca de tempestades! As vírgulas entre as condições devem ser entendidas como AND, pois cada uma das condições deve ser válida para que a conclusão possa ser atingida com confiança. Observe que, apesar de estarmos usando IF. . .THEN, que é familiar na programação, seu uso aqui é diferente. Neste caso, IF. . .THEN não é uma instrução para o computador fazer algo se a condição for verdadeira, mas sim uma declaração de relação entre fatos diferentes.

Para construir um programa que usasse tais regras para resolver problemas, tudo o que seria necessário consistiria em colecionar tantas destas regras quanto possível, ou, em outras palavras, construir um "banco de dados de regras" e, então, para encontrar o significado, ou as conseqüências, de um conjunto de condicões, simplesmente seria necessário pesquisar este banco de dados para encontrar as regras correspondentes a estas condições. Pode haver mais do que uma regra para um conjunto particular de condições. Por exemplo, se o céu estiver negro, este fato estará ligado a ambas as condições exemplificadas atrás, o que permitiria prever chuva, da primeira regra, e uma tempestade, da segunda. Para saber qual teria maior probabilidade, seria necessária informação adicional.

Os programas deste tipo são usualmente chamados de "sistemas especialistas baseados em conhecimentos" e têm recebido

muita atenção da comunidade e do público em geral no momento, como uma das melhores coisas resultantes dos esforços em AI. Tais programas não são absolutamente complicados! O maior problema com todos os sistemas especialistas consiste na seleção da coleção inicial de regras que irá formar o banco de dados, o que é normalmente executado trabalhando-se juntamente com um especialista humano e tentando descobrir que regras ele usa em seu trabalho.

Ao invés de continuar com teoria e maiores explicações, é mais fácil e instrutivo passar diretamente a um sistema especialista simples em BASIC e analisá-lo.

O Programa Aardvark

Toda esta conversa sobre regras de conhecimento e sistemas especialistas pode parecer convincente, mas, será que isto funciona? Para demonstrar o poder destas idéias, o programa apresentado a seguir aprenderá sozinho a ser um especialista em tipos de animais. A razão para usar tipos de animais como exemplo é que este programa em particular já tem uma longa história em uma forma ou outra, sempre sendo apresentado juntamente com um banco de dados de animais. Entretanto, como o programa apreende os dados, ao invés de tê-los embutidos, pode ser usado perfeitamente em outras áreas, tais como diagnóstico de defeitos, simplesmente alterando-se a pergunta. A razão para este ser um sistema especialista muito simples é que ele usa regras do tipo.

IF lista de características de animais THEN é um "nome de animal"

Por exemplo, uma regra seria

IF tem penas e é um predador THEN é uma águia

Para tornar simples a busca das regras corretas e também a inclusão e modificação das regras, todas serão representadas por uma árvore. O uso de uma árvore não é essencial ao método, mas tem várias vantagens práticas e parece ajustar-se naturalmente ao programa.

Para ver como uma árvore pode ser usada para representar um número de regras, considere a Figura 4.1. Começando da primeira questão "tem penas?", pode-se percorrer a árvore respondendo às questões até chegar a um nome de animal. Cada vez que uma questão é respondida o percurso continua pelo ramo correspondente à resposta, ou seja, o ramo da direita para "sim" e o da esquerda para "não", indo à próxima questão.

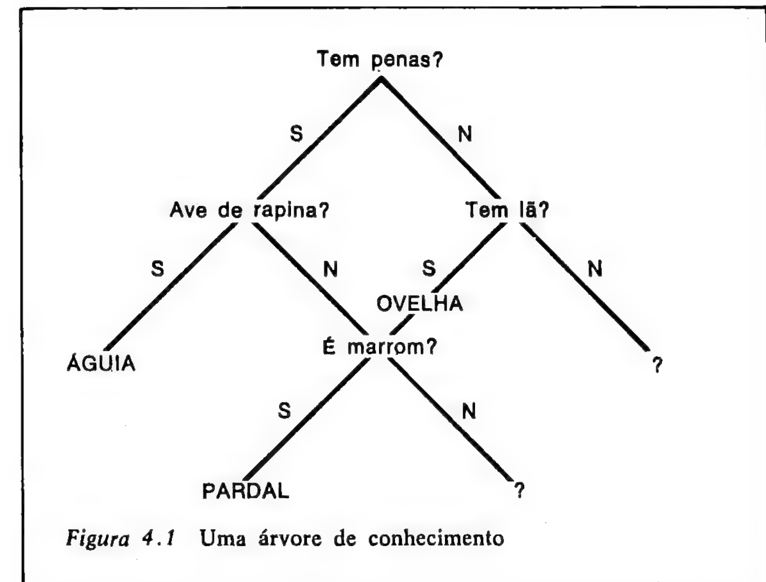


Figura 4.1 Uma árvore de conhecimento

Por exemplo, se a resposta a "tem penas?" for "não", a próxima questão a considerar será "tem lã?". Se a resposta a esta questão for "sim", então o animal é uma ovelha. As regras contidas nesta árvore são

IF penas, rapina THEN águia
IF penas, não rapina, marrom THEN pardal
IF não penas, lã THEN ovelha

Cada uma destas regras é representada na árvore pela trajetória que leva ao nome do animal. A vantagem de armazenar as regras nesta forma é que você pode comparar as condições com

as regras uma a uma, ao invés de todas simultaneamente. Isto permite perguntar ao usuário uma informação de cada vez, ao invés de tudo de uma vez.

Isto tudo está muito bem, mas como é que as regras e a estrutura da árvore aparecem na memória do computador? A resposta é que, a cada vez que o programa Aardvark chega a um "?" na estrutura da árvore, ele não sabe qual é o animal. Então, ele pede ao usuário o nome do animal e também uma questão que poderá ser usada no futuro para identificá-lo. Logo que o computador recebe esta informação, ela é incluída na árvore para uso posterior. Por exemplo, se após apresentar um número de perguntas o programa se encontra no "?" seguinte à questão "é marrom?", perguntará ao usuário "qual é o animal?" e receberá a resposta de que é uma gaivota e que a pergunta que poderá identificá-la é "é um pássaro marinho?". Como resultado disto, a nova questão substituirá o sinal de interrogação, o novo ramo "sim" levará a "GAIVOTA" e o ramo "não" a outro sinal de interrogação.

Uma outra maneira do programa Aardvark aprender é quando chega à resposta errada! Se ele seguir uma trajetória que leve à resposta "PARDAL" e for informado pelo usuário de que o animal não é um pardal, mas uma corruíra, ele evita cometer este engano novamente, solicitando ao usuário uma questão que lhe permita distinguir entre os dois passarinhos. Por exemplo, se a questão fosse "é o menor passarinho marrom da Grã-Bretanha?", então "CORRUIRA" estaria no ramo "sim" e "PARDAL" no ramo "não". Esta nova questão deve substituir a informação "PARDAL" na árvore original. Desta maneira, a árvore "cresce" e se modifica de acordo com as informações que recebe sobre os animais. Somente experimentando o programa é que se pode ver como o computador aprende depressa e como é divertido usar o programa!

O programa está escrito em Microsoft BASIC padrão, como os anteriores, e deve rodar em quase qualquer micro sem modificações, excetuando-se, possivelmente, apenas o ZX81 e o Spectrum, que exigirão algumas pequenas modificações no tratamento dos textos.


```
10 PRINT "AARDVARK"
20 PRINT "VOCE PENSA EM UM ANIMAL E"
30 PRINT "EU VOU DESCOBRI-LO"
40 PRINT "RESPONDA CADA PERGUNTA"
50 PRINT "COM SIM OU NAO"
60 PRINT
70 GOSUB 1000

100 GOSUB 2000
110 PRINT
120 PRINT "PENSE EM OUTRO ANIMAL    "
130 PRINT
140 GOTO 100

1000 DIM Q$(20),R(20),L(20),N$(20)
1010 Q$(1)="TEM PENAS?"
1020 D=1
1030 R(1)=0
1040 L(1)=0
1050 N=0
1060 Q=1
1070 RETURN

2000 X=1
2010 GOSUB 3000
2020 IF A$="S" AND R(X)=0 THEN GOTO 4000
2030 IF A$="N" AND L(X)=0 THEN GOTO 4000
2040 IF A$="S" AND R(X)<0 THEN A=-R(X):GOTO 5000
2050 IF A$="N" AND L(X)<0 THEN A=-L(X):GOTO 5000
2060 IF A$="S" THEN X=R(X)
2070 IF A$="N" THEN X=L(X)
2080 GOTO 2010

3000 PRINT Q$(X)
3010 INPUT A$
3020 A$=LEFT$(A$,1)
3030 IF A$="S" OR A$="N" THEN RETURN
3040 PRINT "EU NAO ENTENDI SUA RESPOSTA"
3050 PRINT "RESPONDA SIM OU NAO"
3060 PRINT "A MINHAS PERGUNTAS - OBRIGADO"
3070 PRINT
3080 GOTO 3000

4000 PRINT "EU NAO CONHECO O ANIMAL"
4010 PRINT "EM QUE VOCE ESTA PENSANDO"
4020 PRINT "COMO ELE SE CHAMA "
4030 INPUT B$
4040 PRINT
4050 PRINT "QUAL OUTRA PERGUNTA EU POSSO FAZER"
4060 PRINT "PARA DISTINGUIR ESSE NOVO ANIMAL"
```

```
4070 INPUT C$
4080 N=N+1
4090 N$(N)=D$
4100 Q=Q+1
4110 Q$(Q)=C$
4120 IF A$="S" THEN R(X)=D+1 ELSE L(X)=D+1
4130 PRINT "PARA ";B$
4140 PRINT "QUAL E A RESPOSTA"
4150 PRINT C$
4160 INPUT D$
4170 D$=LEFT$(D$,1)
4180 IF D$="S" OR D$="N" THEN GOTO 4300
4190 PRINT "RESPONDA SIM OU NAO"
4200 GOTO 4130
4300 D=D+1
4310 IF D$="S" THEN R(D)=-N:L(D)=0
4320 IF D$="N" THEN L(D)=-N:R(D)=0
4330 RETURN

5000 PRINT "E UM(A) ";N$(A);
5010 INPUT B$
5020 IF LEFT$(B$,1)="N" THEN GOTO 6000
5030 IF LEFT$(B$,1)<>"S" THEN PRINT "RESPONDA
     SIM OU NAO":GOTO 5000
5040 PRINT "EU IMAGINAVA !!"
5050 PRINT
5060 RETURN

6000 PRINT "EU DESISTO !"
6010 INPUT "QUAL O ANIMAL ?",B$
6020 PRINT "QUE PERGUNTA PODERIA DIZER"
6030 PRINT "A DIFERENCA ENTRE UM(A) ";N$(A)
6040 PRINT "E SEU/SUA ";B$
6050 INPUT C$
6060 Q=Q+1
6070 Q$(Q)=C$
6080 N=N+1
6090 N$(N)=B$
6100 IF A$="S" THEN R(X)=Q ELSE L(X)=Q
6110 D=D+1
6120 PRINT "PARA ";B$
6130 PRINT "QUAL A RESPOSTA"
6140 PRINT C$
6150 INPUT D$
6160 D$=LEFT$(D$,1)
6170 IF D$="S" THEN R(D)=-N:L(D)=-A
6180 IF D$="N" THEN L(D)=-N:R(D)=-A
6190 PRINT
6200 RETURN
```


Os detalhes do programa não são difíceis de entender. A estrutura de sub-rotinas é a seguinte:

10	Instruções introdutórias
100	Ciclo do programa principal — executado uma vez para cada animal
1000	Sub-rotina de início
2000	Faz a pergunta e processa a resposta
3000	Rotina de entrada de resposta
4000	Não conhece o animal, pede novo animal e questão identificadora
5000	Relata qual o animal encontrado e testa a correção
6000	Resposta errada, pede novo animal e questão identificadora

A única outra informação importante é que a estrutura da árvore é representada por dois *arrays*, L e R, correspondentes aos ramos esquerdos e direitos seguindo a cada questão. Alcançado um elemento do qual o *array* R ou o L contenha zero, isto significa que o final da árvore foi encontrado e, portanto, não sabe qual é o animal. Alcançado um elemento de um dos *arrays* com um número negativo, isto significa que foi achado um possível candidato para o animal. Os nomes dos animais são armazenados no *array* N$ e o índice do animal que foi achado no *array* R ou no L, como um número negativo. Assim, se L(X) for negativo, o nome do animal estará em N$(—L(X)). Se, finalmente, o valor armazenado em R ou L for positivo e diferente de zero, representará o índice da próxima questão que deve ser proposta ao usuário e também o índice do próximo elemento tanto de R como de L, que deve ser examinado como resultado da resposta à questão. Finalmente, as questões são armazenadas no *array* Q$.

A árvore, inicialmente, contém uma questão e nenhum nome de animal, estando a questão em Q$(1) e R(1) e L(1) contendo zero (zero representa o sinal de interrogação no diagrama da árvore da Figura 4.1). Se a resposta à primeira questão for "sim", R(1) será examinado e, como seu valor é zero, o programa pedirá um nome de animal e uma nova questão. O novo nome será armazenado em N$(1) e a questão em Q$(2). R(1) é mudado para 2, de modo que a questão Q(2) será proposta em resposta a um "sim" na questão 1 e R(2) e L(2) são alterados de modo a um deles conter —1, indicando que o nome do animal está em N$(1) e o outro é zerado para indicar

que são necessários outro nome e outra questão. A Figura 4.2 dá uma amostra da saída do programa Aardvark.

Aperfeiçoamento do Programa Aardvark

O programa Aardvark é simples, mas chega efetivamente a uma conclusão a partir de um conjunto de condições e, de fato, aprende tanto de sua ignorância quanto de seus erros. A maneira em que o conhecimento cresce é desorganizada, mas é flexível. Por exemplo, um animal pode aparecer em mais de um lugar na árvore, permitindo que sua "definição" seja feita por diferentes conjuntos de condições. Seu maior problema é que a ordem em que as questões são propostas é governada simplesmente pela ordem em que são aprendidas e esta usualmente não é a melhor ordem para sua proposição. Se um humano estivesse executando o trabalho de Aardvark, sua primeira questão seria escolhida de modo a fornecer o máximo de informação. Por exemplo, perguntando "tem penas?" inicialmente, o restante da busca pode ser imediatamente reduzida aos pássaros ou aos não-pássaros. Poderia ser incluída uma nova seção no programa que examinaria a árvore e encontrasse o melhor arranjo possível para as questões, mas isto está além do objetivo deste livro.

Conforme já foi sugerido, substituindo a primeira questão por algo tal como "o defeito foi elétrico?", o programa poderia ser transformado em outro que descobrisse defeitos. Obviamente, dizer qual é o defeito é somente parte da solução. Seria necessário, neste novo programa, incluir também instruções sobre o que fazer juntamente com o nome de cada defeito. Outras

```
AARDVARK
VOCE PENSA EM UM ANIMAL E
EU VOU DESCOBRI-LO
RESPONDA CADA PERGUNTA
COM SIM OU NAO

TEM PENAS?
? nao
EU NAO CONHECO O ANIMAL
EM QUE VOCE ESTA PENSANDO
COMO ELE SE CHAMA
? cachorro
```



```
QUAL OUTRA PERGUNTA EU POSSO FAZER
PARA DISTINGUIR ESSE NOVO ANIMAL
? ele late?
PARA CACHORRO
QUAL E A RESPOSTA
ELE LATE
? sim

PENSE EM OUTRO ANIMAL

TEM PENAS?
? nao
ELE LATE?
? nao

EU NAO CONHECO O ANIMAL
EM QUE VOCE ESTA PENSANDO
COMO ELE SE CHAMA
? boi

QUAL OUTRA PERGUNTA EU POSSO FAZER
PARA DISTINGUIR ESSE NOVO ANIMAL
? tem chifres?
PARA BOI
QUAL E A RESPOSTA
TEM CHIFRES
? sim

PENSE EM OUTRO ANIMAL

TEM PENAS?
? nao
ELE LATE?
? nao
TEM CHIFRES?
? sim
E UM(A) BOI ? nao
EU DESISTO !
QUAL O ANIMAL ? vaca
QUE PERGUNTA PODERIA DIZER
A DIFERENCA DE UM(A) BOI
E SEU/SUA VACA
? e femea
PARA VACA
QUAL A RESPOSTA
E FEMEA
? sim
```

Figura 4.2 Amostra da saída do programa Aardvark. Para maior clareza, as respostas são dadas em letras minúsculas

sugestões de aperfeiçoamentos do programa Aardvark poderiam ser a adição de facilidades para

guardar e ler informações em uma árvore existente
achar um jeito de imprimir toda a árvore para análise
permitir que o usuário solicite a descrição de qualquer animal constante da árvore.

O programa Aardvark não é tão sofisticado como os sistemas especialistas totalmente comerciais, particularmente no fato de não prever conclusões incertas.

Cuidado com Excessiva Confiança

A idéia de um programa de computador que executa raciocínios usando regras do tipo IF condição THEN conclusão, parece muito natural e promissora, até que se comece a pensar no tipo de julgamento que os humanos fazem. Quase nunca uma decisão pode ser alcançada com certeza absoluta, sendo mais comum chegar-se a conclusões do tipo "acho que o problema poderia ser" ou "pode ser que...", do que "tenho a certeza de que..." ou "é...". Até agora, todas as regras tipo IF...THEN consideradas admitiram que a resposta a qualquer questão seja conhecida com absoluta certeza e, conseqüentemente, que a presença de qualquer condição dada seja sempre um sinal certo de que a conclusão respectiva deve ser tirada. Obviamente, é claro que isto nem sempre acontece! Um importante componente do raciocínio humano que não foi considerado é a incerteza.

Há duas maneiras de tratar as incertezas no raciocínio. A primeira é tradicional, sendo baseada em probabilidades. A segunda é bastante recente e ainda não muito desenvolvida, de forma que deixa campo para experimentação. Entretanto, é importante compreender que há dois tipos diferentes de incerteza presentes no raciocínio, sendo o primeiro simplesmente não ter muita certeza da condição. Por exemplo, um animal em movimento pode ser observado por um instante tão pequeno, que não dê para ver se tem cauda ou não. Este tipo consiste, portanto, em não se ter certeza da evidência e é fácil de tratar nos programas. O segundo tipo de incerteza é aquele no qual a evidência é clara, ou seja, existe absoluta certeza dos fatos, mas não existe uma conexão clara entre estes e a conclusão que se

pretende tirar. Por exemplo, o tempo estar úmido somente permite concluir que há boas chances de haver uma tempestade. Esta forma de incerteza é conhecida por "incerteza na inferência" e não pode ser ignorada como um importante componente do raciocínio.

Probabilidade

É muito familiar a idéia da probabilidade de algo ocorrer, mas poucos têm uma idéia clara de que, quanto mais alta a probabilidade, maior a possibilidade de que algo ocorra. Uma probabilidade de zero corresponde à certeza de que um evento não ocorrerá e, semelhantemente, uma probabilidade de um à certeza de que um evento ocorrerá. Assim, probabilidades com valor zero ou um correspondem a certezas, enquanto que valores intermediários correspondem a incertezas. (Observe a semelhança entre probabilidade e lógica Booleana.) A melhor interpretação da probabilidade é aquela que usa o número de vezes que algo ocorre, tal como, por exemplo, quando se joga uma moeda que não seja viciada um grande número de vezes, deverá dar cara metade das vezes. Neste sentido, dizer que "a probabilidade de dar cara é de 0,5" é uma afirmativa sobre a proporção de vezes que se espera que dê cara em um grande número de jogadas. Esta idéia é usualmente generalizada para uma interpretação de que a probabilidade de outros eventos seja a proporção de vezes que deverão ocorrer a longo prazo.

Esta parece uma maneira sólida e realista de compreender probabilidade e é, geralmente, considerada como a melhor. Considere, entretanto, o caso em que se deseja saber qual a probabilidade de algo ser verdadeiro ou falso, quando fica difícil ver como a idéia de que o número de vezes que algo é verdadeiro ou falso, a longo prazo, pode ser usada como interpretação da probabilidade. Por exemplo, "qual você acha que é a probabilidade de haver vida em outros planetas?" é uma questão que pode até ser respondida, mas é difícil relacionar a probabilidade dada como resposta com o número de vezes que o fato de haver vida em outros planetas ocorra a longo prazo! Mesmo a imaginação mais louca terá dificuldade em trabalhar com mais do que um universo, para poder repetir o evento! Há muitas maneiras de evitar esta dificuldade, sendo a mais fácil

delas simplesmente abandonar a interpretação direta de probabilidade como uma indicação de quantas vezes se espera que algo ocorra. É claro que, feito isto, probabilidade somente pode ser usada como uma vaga medida da certeza relativa sobre alguma coisa, não podendo o resultado ser verificado da mesma maneira que a proporção de vezes que um evento ocorre. Neste sentido, probabilidade fica sendo uma noção subjetiva, não havendo nada que a recomende mais do que qualquer outra medida de incerteza, logo que seu significado físico seja abandonado.

Ao invés de usar probabilidade, poder-se-ia usar uma outra medida da incerteza, com um conjunto de regras diferente para combinar tais medidas e, já que tudo é subjetivo, ninguém poderia reclamar! O importante é que, enquanto probabilidade pode ser exatamente a maneira adequada para exprimir nossa incerteza sobre alguns eventos no mundo, não é necessariamente a melhor maneira de exprimir a maneira na qual os humanos se sentem "incertos" sobre algo. Para aqueles que estiverem vendo com ceticismo o uso de outras maneiras de tratar a incerteza, que não a probabilidade, será discutida mais adiante neste capítulo uma das alternativas, a lógica nebulosa.

Leis para o Pensamento Incerto?

Apesar de haver restrições filosóficas sobre o uso de probabilidades em certas áreas do conhecimento humano e do raciocínio, este é ainda o sistema que mais se usa. Por esta razão, vale a pena examinar como seria possível adicionar probabilidades aos sistemas especialistas do tipo de Aardvark. Para fazer isto, é necessário algum conhecimento básico da teoria das probabilidades. A probabilidade de um evento x é normalmente denotada $P(x)$, que deve ser entendido como uma representação taquigráfica de "a probabilidade de x". Por exemplo, P(cara) representa a probabilidade de se obter cara quando uma moeda for jogada, cujo valor, se a moeda não for viciada, é igual a 0,5. Uma idéia um pouco mais complicada, mas muito útil, é a de uma "probabilidade condicional", que é usualmente denotada $P(A|B)$ e lido como "a probabilidade de que A aconteça, desde que B tenha acontecido". (Será mais fácil ler o restante deste capítulo se esta expressão for reduzida para "a probabi-

lidade de A, dado B", ou seja, se P for entendido como sendo "a probabilidade" e a barra vertical como "dado".) Assim, por exemplo, P(chuva) é simplesmente a probabilidade da chuva, mas P (chuva|nuvens negras) é a probabilidade de chuva quando existirem nuvens negras no céu. Em outras palavras, uma probabilidade condicional é a probabilidade de algo acontecer, após incluído no raciocínio algum conhecimento já disponível.

É fácil ver que as probabilidades condicionais são semelhantes a regras IF. . . THEN incertas. Havendo certeza absoluta sobre algo, pode-se escrever regras do tipo

IF nuvens negras no céu THEN chuva

enquanto que, se uma certa incerteza for admitida, a regra ficaria melhor como

$$P(\text{chuva}|\text{nuvens negras}) = 0{,}9$$

que dá uma indicação razoável da segurança da conclusão tirada na primeira regra.

Probabilidades condicionais são um pouco mais difíceis de usar do que indicado no exemplo atrás, devido ao problema da determinação dos valores que deverão efetivamente ser usados. Há, entretanto, uma relação matemática muito útil entre $P(A|B)$ e $P(B|A)$:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Usando esta simples equação, é possível obter facilmente P(chuva|nuvens negras) desde que P(nuvens negras|chuva) seja conhecido:

$$P(\text{chuva}|\text{nuvens negras}) = \frac{P(\text{nuvens negras}|\text{chuva})P(\text{chuva})}{P(\text{nuvens negras})}$$

Como P(nuvens negras|chuva) pode ser admitido como sendo igual a 1 (ou seja, *sempre* que chove há nuvens negras no céu), isto dá

$$P(\text{chuva}|\text{nuvens negras}) = \frac{P(\text{chuva})}{P(\text{nuvens negras})}$$

o que é muito razoável, considerando-se o significado desta última equação. $P(\text{chuva})$ é simplesmente a proporção de tempo que chove, enquanto que $P(\text{nuvens negras})$ é a proporção de tempo que aparecem nuvens negras no céu, esteja ou não chovendo.

A idéia de trocar $P(B|A)$ por $P(A|B)$ não somente é muito útil na prática, mas também forma a base do método padrão de raciocinar com probabilidades. A equação ligando $P(B|A)$ a $P(A|B)$ é chamada de Teorema de Bayes por ser devida ao Reverendo Thomas Bayes e, assim, seu uso no raciocínio é usualmente denominado "inferência Bayesiana". A melhor estimativa de alguém, sentado em um quarto sem janelas e sem nenhuma idéia de como está o tempo lá fora, sobre se vai chover ou não deveria ser simplesmente $P(\text{chuva})$, ou seja, a proporção de tempo que normalmente chove. Neste sentido, $P(\text{chuva})$ exprime quanto se acredita que pode estar chovendo na ausência de qualquer outro conhecimento. Dada, entretanto, a informação de que há nuvens negras no céu, seria necessário rever a esperança de que chova ou não, de forma a levar este novo dado em consideração. Usando

$$P(\text{chuva}|\text{nuvens negras}) = \frac{P(\text{nuvens negras}|\text{chuva})P(\text{chuva})}{P(\text{nuvens negras})}$$

é fácil converter a esperança anterior $P(\text{chuva})$, antes da nova informação de que há nuvens negras no céu, para a nova esperança, $P(\text{chuva}|\text{nuvens negras})$. Se novas informações chegarem ao quarto fechado, por exemplo, de que está trovejando, o mesmo método poderia ser usado para atualizar novamente a esperança de que chove:

$$P(\text{chuva}|\text{trovão, nuvens negras}) = \frac{P(\text{chuva}|\text{nuvens negras})\ P(\text{trovão}|\text{chuva})}{P(\text{trovão})}$$

e assim por diante. Cada nova peça de informação seria usada para modificar a esperança anterior em uma nova esperança e tudo o que é necessário fazer para isto é multiplicar a probabilidade que representa a esperança anterior pela constante que indica de quanto a nova evidência afeta, a favor ou contra, a

esperança anterior. No caso da chuva e do trovão, por exemplo, a constante é $P(\text{trovão}|\text{chuva})/P(\text{trovão})$. Em certos casos extremos, a nova evidência pode ser tão avassaladora, que a esperança se transforma em certeza. Por exemplo, se *nunca* ocorressem nuvens negras sem chuva e chuva *nunca* ocorresse sem nuvens negras, então $P(\text{nuvens negras}|\text{chuva})$ seria igual a 1 e $P(\text{nuvens negras})$ seria igual a $P(\text{chuva})$, já que ambos ocorreriam sempre juntos. Usando estes valores (que obviamente não são verdadeiros!) na equação anterior, vem:

$$P(\text{chuva}|\text{nuvens negras}) = \frac{1}{P(\text{chuva})} P(\text{chuva}) = 1$$

o que, no mínimo, mostra que isto funciona de acordo com o senso comum.

Este método poderia ser adicionado ao Aardvark para produzir um sistema especialista que processasse regras IF... THEN incertas. A cada ponto de decisão na árvore, seria incluída uma nova peça de informação que poderia ser usada para atualizar a probabilidade de cada conclusão. Seja, por exemplo, uma árvore de conhecimento consistindo de somente três animais: leão, tigre e onça. No início do programa, a "confiança" que o computador poderia ter de que um dos animais seria o que o usuário imaginou, seria simplesmente $P(\text{leão})$, $P(\text{tigre})$ e $P(\text{onça})$. (Para facilitar a compreensão desta afirmativa, é bom comparar com o exemplo anterior, da previsão do tempo feita de um quarto fechado, onde, na ausência de qualquer informação adicional, a esperança de que estivesse chovendo era simplesmente de $P(\text{chuva})$.) Cada vez que uma questão é respondida pelo usuário, ou que mais informações são dadas ao programa, este atualizará sua confiança de que cada um dos animais possa ser a solução correta. Por exemplo, dizer ao programa que o animal corre rápido não o ajudaria a decidir que o animal é uma onça, pois leões e tigres também correm rápido! Ao contrário, o programa poderia sim usar esta informação para aumentar a probabilidade de o animal ser uma onça e diminuir, correspondentemente, as probabilidades de ser um leão ou um tigre. Entretanto, recebendo uma informação que fizesse a probabilidade de ser um dos animais igual a 1, o programa daria imediatamente esta decisão. Em muitos casos,

porém, mesmo após receber tudo o que usuário sabe sobre o animal, o programa ainda não poderia decidir com certeza qual seria a resposta, sendo obrigado, nestes casos, a dar uma resposta com o animal de máxima probabilidade, ou mesmo com uma lista de animais de alta probabilidade.

Obviamente, esta idéia pode ser generalizada para uso em aplicações mais sérias e úteis. Por exemplo, no diagnóstico de defeitos de automóveis, o sistema especialista iniciaria com uma lista de defeitos possíveis e suas probabilidades. Conteria também uma série de regras de probabilidades, tais como P(defeito no carburador|cheiro de combustível) = 0,8 ou P(furo no tanque de combustível|cheiro de combustível) = 0,9, que seriam usadas para atualizar as probabilidades iniciais à medida que a evidência encontrada fosse sendo apresentada ao programa. O resultado final não será necessariamente um diagnóstico claro e certo, mas possivelmente uma lista de defeitos possíveis com suas respectivas probabilidades.

Não é incluído aqui um programa BASIC que utilize inferência incerta simplesmente devido ao fato de que o sucesso de um tal programa depende muito da qualidade do banco de conhecimento. A construção de um bom sistema especialista leva muito tempo, mas os princípios de sua operação não são difíceis de entender.

Evidência Incerta

A seção anterior discutiu o mais difícil aspecto da incerteza, mas não deu, realmente, uma pista para o que fazer quando a evidência for incerta. A resposta para isto é muito simples, mas muito pouco satisfatória. Incluindo-se em um sistema especialista uma questão suplementar tal como "Quão seguro você está (numa escala de 0 a 1) de sua última resposta?" seria possível reunir estimativas da segurança com que é dada cada informação que será usada para basear a conclusão. Tratando tais estimativas como probabilidades, fica bastante complicada a maneira correta de utilizá-las para estimar a incerteza na conclusão, dependendo do conhecimento das relações existentes entre as várias peças de informação. Como tais relações são usualmente desconhecidas, é comum admitir que sejam inexistentes, o que implica que a estimativa correta da incerteza da conclusão

seja obtida pela multiplicação de todas as probabilidades. (Isto decorre de um teorema da teoria das probabilidades, que diz que, se vários eventos são independentes, a probabilidade de todos ocorrerem simultaneamente é igual ao produto de suas probabilidades individuais.) Isto parece muito razoável até que se observe que 0,5 vezes 0,5 que dá 0,25, já é bem pequeno e que, mesmo começando com probabilidades altas, tais como 0,9 (quase uma certeza), a probabilidade da conclusão acabará sendo muito pequena, se houver muita evidência! O produto das probabilidades dá uma estimativa muito conservadora da confiança que se pode ter na conclusão, pois as relações entre os fatos tomados como evidência estarão todas sendo desprezadas!

Na realidade, não há nenhuma solução aceitável, até o momento, para este problema, e a maioria dos sistemas especialistas tem sua maneira própria de tratá-lo. Entretanto, há um método que parece promissor, pois ajusta-se ao tópico considerado na próxima seção, lógica nebulosa, e consiste em, ao invés de multiplicar todas as probabilidades, a confiança na conclusão é estimada tomando-se o mínimo das probabilidades.

Alternativas para Probabilidades: Lógica Nebulosa

Deve ter sido observado que, embora o tratamento inicial tenha sido exato, usando probabilidades de maneira precisa, a seção sobre pensamento incerto introduziu a idéia de que o que interessa não é tanto a exatidão dos valores dados (ou estimados), mas sim a maneira na qual mostram aproximadamente a forma em que uma peça de evidência fortalece ou enfraquece uma conclusão. A última seção mostrou que é melhor não usar métodos probabilísticos para estimar a confiança que se pode ter em uma conclusão. De fato, logo que se comece a trabalhar com estimativas de confiança e com a evidência, nada exige que se utilize probabilidades. Um sistema alternativo consiste em utilizar a lógica nebulosa, que é fácil de explicar do ponto de visa dos adjetivos. Considere a palavra "alto" e a maneira de usá-la. Há sempre uma certa incerteza em sua aplicação: não é que alguém tenha uma determinada probabilidade de ser alto, mas sim que pode ser mais ou menos alto. Alguém com 2m de altura é claramente alto, mas e alguém com 1,80m, ou com

1,75m, ou mesmo com 1,70m? Fica claro que "ser alto" não é algo que possa ser facilmente decidido por uma decisão do tipo sim/não. A lógica tradicional usa um para indicar verdadeiro e 0 para falso, sem valores intermediários. Na lógica nebulosa, pode-se usar "graus de verdade" entre 0 e 1, novamente com 0 indicando falso e 1 indicando verdadeiro, sendo que os valores intermediários representam algo "mais ou menos verdadeiro ou falso". Por exemplo, a resposta à questão "ele é alto?" pode não ser um valor 0 ou 1, mas algo intermediário, tal como 0,5 ou 0,8.

Esta idéia de usar valores lógicos intermediários pode parecer meio estranha no começo, mas é fácil de acostumar-se com ela. As operações fundamentais da lógica tradicional, AND, OR e NOT, podem ser facilmente generalizadas para valores lógicos nebulosos:

A AND B transforma-se em MIN(A,B)
A OR B transforma-se em MAX(A,B)
NOT(A) transforma-se em 1 — A

independentemente de A e B serem valores Booleanos tradicionais ou valores lógicos nebulosos. Em outras palavras, mesmo em lógica tradicional, A AND B é equivalente a MIN(A,B):

A	*B*	*A AND B*	*MIN(A,B)*
0	0	0	0
0	1	0	0
1	0	0	0
1	1	1	1

Isto significa que é possível trabalhar com lógica nebulosa, sendo que se os valores lógicos, por acaso, forem exatamente 0 ou 1, não ocorre nada diferente da álgebra Booleana; obviamente a lógica tradicional deve obrigatoriamente ser um subconjunto de qualquer novo tipo de raciocínio que se planeje utilizar. Esta introdução simples pode agora prosseguir para desenvolver o total da lógica, agora trabalhando com valores lógicos nebulosos.

Muitos aspectos da lógica tradicional não se transportam tão facilmente para a lógica nebulosa, o que implica não ser possível simplesmente substituir a lógica tradicional pela

nebulosa em alguma aplicação dada, sendo que, até o momento, ninguém conseguiu inserir em um sistema especialista a lógica nebulosa de maneira convincente, mas não se assuste: este assunto ainda está na infância!

Um dos interessantes usos da lógica nebulosa é a descrição de idéias humanas para computadores. Por exemplo, o adjetivo "alto" que trouxe tanta dificuldade na discussão anterior poderia ser facilmente descrito por um gráfico indicando o valor lógico para várias alturas. Esta descrição é tão fácil de dar ao computador quanto a um outro humano. Da mesma forma que a descrição simples de palavras tais como "alto", a lógica nebulosa pode ser usada para dar ao computador regras vagas, tais como IF "a caldeira estiver quente" THEN "reduza o calor", ou IF "a caldeira estiver muito quente", THEN "reduza muito o calor". As palavras "quente", "muito quente" e "muito" são muito difíceis de descrever a um computador, a menos que se use lógica nebulosa. Esta é uma idéia que já levou a aplicações reais, tais como controlar uma máquina a vapor ou um forno de cimento! Observe, agora, que voltamos ao início! Os Capítulos 2 e 3 propuseram a idéia da heurística, opondo-a aos algoritmos. Heurísticas são regras vagas, que *tendem* a dar uma solução. Qual seria a melhor maneira para descrevê-las que a lógica nebulosa?

A Condição e a Conclusão

Antes de abandonar o assunto de raciocinar usando regras do tipo IF. . .THEN, é interessante mostrar sua grande generalidade. Pode ser surpreendente verificar que é possível construir um sistema especialista que aprenda a atuar em jogos tais como os descritos nos Capítulos 2 e 3, mas isto não invalida tudo o que foi visto (heurísticas, árvores de movimentos, funções de avaliação e a estratégia minimax), pelo contrário, consolida todos estes assuntos num enfoque unificado. O poder das regras tipo IF condição THEN conclusão deriva exatamente do fato de que os termos "condição" e "conclusão" são vagos. A "condição" poderia perfeitamente ser uma posição em um jogo e a "conclusão" a próxima jogada, ao invés de uma questão e uma resposta, tal como em Aardvark. O fato de os termos serem vagos constitui também um importante problema para a imple-

mentação de um sistema de raciocínio mais ambicioso. É muito difícil inserir em um programa um mecanismo que processe toda a enorme gama de condições e conclusões que o sistema humano de raciocínio parece capaz de tratar. Por enquanto, cabe ao programador de AI a tarefa de resolver o problema de encontrar maneira de representar condições e implementar conclusões, antes que um sistema especialista possa começar a resolver problemas aplicados usando as regras IF. . . THEN.

Aprendendo o Jogo da Velha

Conforme já foi dito, o jogo da velha é muito bom como ilustração dos métodos de AI, pois é suficientemente simples para permitir que os programas BASIC correspondentes sejam curtos, enquanto que é suficientemente complicado para demonstrar as técnicas discutidas. Assim, o jogo da velha será novamente usado para ilustrar as idéias envolvidas no uso de um sistema especialista em jogos.

A primeira característica necessária para aprender a fazer alguma coisa é uma boa memória, onde possa ser guardada a informação sobre o que funcionou e o que não funcionou. No caso do jogo da velha, será necessário lembrar movimentos que levaram à derrota, para não repeti-los mais. Um tabuleiro de jogo da velha tem nove posições, cada uma das quais pode conter um espaço, um X ou um O. Codificando o espaço por 0, o X por 1 e o 0 por 2, cada posição em uma partida poderá ser registrada como uma seqüência de algarismos consistindo apenas de 0, 1 e 2. Por exemplo, 0000000010 corresponderia a um tabuleiro com oito espaços e um X. É perfeitamente possível armazenar as posições no tabuleiro registrando os algarismos como texto, mas, num esforço para produzir um programa que funcione em uma grande variedade de máquinas, é melhor registrá-los como números em um *array*. Este método somente funcionará em máquinas cujas versões do BASIC possam armazenar números de nove algarismos. Para verificar se sua máquina pode fazer isto, tente:

```
10 A = 333333333
20 PRINT A
```


Se o resultado impresso for exatamente igual ao número armazenado em A na linha 10 do programa, seu BASIC pode trabalhar com números de nove algarismos. Não sendo este o caso, será necessário recorrer a variáveis inteiras (se o BASIC usado dispuser delas), ou ainda reescrever todo o programa para usar textos.

O programa apresentado a seguir usa a idéia de armazenar as posições do tabuleiro como números de nove algarismos no *array* M. A sub-rotina 5000 converte a situação atual, armazenada no *array* bidimensional B em um único número, que será armazenado em P. O programa começa oferecendo ao jogador humano uma jogada (sub-rotina 3000) e, em seguida, armazenando em M a situação resultante do tabuleiro. Isto é executado pela sub-rotina 4000, que chama a sub-rotina 5000 para codificar a situação do tabuleiro em um único número P, testa se esta situação já está armazenada em M usando a sub-rotina 6000 e, caso não esteja, chama a sub-rotina 6500 para fazê-lo.

O programa desenvolvido até agora lembra as posições do tabuleiro à medida que o jogo progride. Em seguida, será necessário dar-lhe condições de escolher seus movimentos e, principalmente, de aprender a jogar. Este próximo estágio de desenvolvimento admite que a resposta ao movimento do jogador humano deva depender exclusivamente da situação atual do tabuleiro, o que significa que é possível imaginar a ação de jogar o jogo da velha como

IF posição atual for X, THEN execute o movimento Y

Ou seja, é aqui que as regras IF. . . THEN dos sistemas especialistas entram em ação! Em paralelo com o *array* M, deve haver um outro *array* R para armazenar as respostas, ligados ambos pelo índice, ou seja, se a posição atual do tabuleiro estiver armazenada em M(I), a resposta correspondente estará armazenada em R(I). A melhor maneira de armazenar as respostas consiste em usar a mesma codificação das posições do tabuleiro. Como o programa ainda não aprendeu nada do jogo da velha, sua resposta pode ser jogar simplesmente na primeira posição livre do tabuleiro, e é isto o que é feito. À medida que vai sendo obtida cada nova posição do tabuleiro, esta é armazenada em M e em R pela sub-rotina 4000, sendo em seguida

chamada a sub-rotina 8000 para executar um movimento na primeira posição livre.

Obviamente, este jogo quase ao acaso pode ser facilmente vencido e, assim, o programa armazena os detalhes das posições do tabuleiro e de suas próprias jogadas em resposta, mas sempre perde! O próximo estágio de desenvolvimento do programa consiste, então, em eliminar as regras IF. . . THEN inconvenientes, o que é surpreendentemente fácil de fazer. Se o programa acabou de perder um jogo, o último movimento que ele fez foi claramente um movimento muito inconveniente. O melhor a fazer, então, é buscar na memória a regra IF. . . THEN que perdeu o jogo, manter sua parte IF. . . mas alterar o que segue ao THEN. Sendo os movimentos em resposta feitos sempre para a primeira posição livre do tabuleiro, e, sendo estas representadas por O em R, tudo o que é necessário fazer para impedir que a posição inconveniente seja usada é trocar o O por outro número, por exemplo, um 3, nesta posição, o que é feito pela sub-rotina 7500. Assim, na próxima vez que a posição do tabuleiro, que levou à derrota, aparecer, o programa usará uma outra jogada, que, se também levar à derrota, será igualmente eliminada com um 3. Este processo de troca da jogada continuará até que seja encontrada uma jogada vencedora.

O programa dado a seguir é muito fácil de vencer a princípio, mas à medida que for sendo usado, vai aprendendo a não perder, o que, no jogo da velha, é quase tão bom como aprender a ganhar!

```
 10 REM APRENDENDO A JOGAR
 20 REM JOGO DA VELHA
 30 GOSUB 1000
 40 L=0:M=0
 50 GOSUB 2000
 55 M=M+1
 60 GOSUB 3000
 65 GOSUB 2000
 66 GOSUB 8500
 70 IF L<>0  THEN GOTO 9000
 80 GOSUB 4000
 85 GOSUB 2000
 90 GOSUB 8500
 95 M=M+1
100 IF L<>0 THEN GOTO 9000
110 GOTO 60
```



```
1000 DIM M(100),R(100)
1010 DIM B(3,3)
1020 N=100
1030 RETURN

2000 FOR I=1 TO 3
2010 FOR J=1 TO 3
2020 IF B(I,J)=0 THEN PRINT ".";
2030 IF B(I,J)=1 THEN PRINT "X";
2040 IF B(I,J)=2 THEN PRINT "O";
2060 NEXT J
2070 PRINT
2080 NEXT I
2090 RETURN

3000 PRINT
3010 PRINT "SEU MOV. (LIN,COL)";
3020 INPUT R,C
3030 IF R>3 OR R<1 THEN GOTO 3010
3040 IF C>3 OR C<1 THEN GOTO 3010
3050 IF B(R,C)=0 THEN GOTO 3080
3060 PRINT "JA ESTA OCUPADO !"
3070 GOTO 3010
3080 B(R,C)=1
3090 RETURN

4000 PRINT
4005 GOSUB 5000
4010 GOSUB 6000
4030 IF S>N THEN GOTO 4500
4040 IF F=0 THEN GOSUB 6500
4050 IF F=1 THEN GOSUB 7000
4060 RETURN

4500 PRINT "ACABOU MINHA MEMORIA"
4510 STOP

5000 P=0
5010 FOR I=1 TO 3
5020 FOR J=1 TO 3
5030 P=P*10+B(I,J)
5040 NEXT J
5050 NEXT I
5060 RETURN

6000 S=1
6010 F=0
6020 IF S>N THEN RETURN
6030 IF M(S)=0 THEN RETURN
6040 IF M(S)=P THEN F=1:RETURN
6050 S=S+1
6060 GOTO 6020
```

```
6500 M(S)=P
6510 Q=P
6520 R(S)=P
6530 GOSUB 8000
6540 RETURN

7000 Q=R(S)
7010 GOSUB 8000
7020 RETURN

7500 Q=R(S)
7510 X=0:Y=0
7520 A=Q-INT(Q/10)*10
7530 Q=INT(Q/10)
7540 X=X+1
7550 IF A<>0 THEN GOTO 7520
7560 Y=4
7570 IF X=1 THEN GOTO 7610
7580 FOR I=1 TO X-1
7590 Y=Y*10
7600 NEXT I
7610 R(S)=R(S)+Y
7620 RETURN

8000 X=0
8010 A=Q-INT(Q/10)*10
8020 Q=INT(Q/10)
8030 X=X+1
8040 IF A=0 THEN GOTO 8100
8050 IF X=9 THEN L=1:RETURN
8060 GOTO 8010

8100 X=9-X
8110 R=INT(X/3)
8120 C=X-R*3
8130 B(R+1,C+1)=2
8140 RETURN

8500 FOR I=1 TO 3
8510 IF B(I,1)=1 AND B(I,2)=1 AND B(I,3)=1
     THEN L=1
8520 IF B(I,1)=2 AND B(I,2)=2 AND B(I,3)=2
     THEN L=2
8530 IF B(1,I)=1 AND B(2,I)=1 AND B(3,I)=1
     THEN L=1
8540 IF B(1,I)=2 AND B(2,I)=2 AND B(3,I)=2
     THEN L=2
8550 NEXT I
8560 IF B(1,1)=1 AND B(2,2)=1 AND B(3,3)=1
     THEN L=1
8570 IF B(1,1)=2 AND B(2,2)=2 AND B(3,3)=2
     THEN L=2
```



```
8580 IF B(1,3)=1 AND B(2,2)=1 AND B(3,1)=1
     THEN L=1
8590 IF B(1,3)=2 AND B(2,2)=2 AND B(3,1)=2
     THEN L=2
8600 RETURN

9000 IF L<>1 THEN GOTO 9050
9010 PRINT "VOCE VENCEU ..."
9020 PRINT "MAS EU VOU APRENDER COM MEU ERRO!"
9040 GOSUB 7500
9045 GOTO 9100
9050 IF M=9 THEN GOTO 9100
9060 PRINT "EU VENCI"
9100 PRINT "JOGAR NOVAMENTE ?";
9110 INPUT A$
9120 IF LEFT$(A$,1)="N" THEN STOP
9130 FOR I=1 TO 3
9140 FOR J=1 TO 3
9150 B(I,J)=0
9160 NEXT J
9170 NEXT I
9180 GOTO 40
```

Com a descrição feita, a compreensão dos detalhes do programa é muito simples, sendo sua estrutura a seguinte:

10	programa principal
1000	início
2000	imprime o tabuleiro
3000	movimento do jogador humano
4000	movimento do computador (usa outras sub-rotinas também)
5000	codifica a posição do tabuleiro em um único número
6000	determina a posição atual em M
6500	não encontrou a posição, armazena-a em M e em R
7000	encontrou a posição, obtém a resposta de R
7500	perdeu o jogo, portanto remove o primeiro O na jogada perdedora
8000	determina a primeira posição livre, para executar o movimento do computador
8500	alguém já ganhou?
9000	final da partida

Aardvark e o Jogo da Velha

Existem semelhanças e diferenças entre o programa Aardvark e o sistema especialista do Jogo da Velha. Ambos usam a regra

tipo IF "condição" THEN "conclusão", sendo, entretanto, as "condições" e "conclusões" diferentes em cada caso. No programa Aardvark, a "condição" é uma lista possivelmente longa de atributos, ligados por cláusulas AND e, no caso do jogo da velha, a "condição" é uma posição particular do tabuleiro, representada num código numérico. Não se deve, entretanto, perder de vista o fato de que estas condições são do mesmo tipo. Por exemplo, a posição do tabuleiro poderia ser codificada como "um X no canto direito ao alto AND um X no canto direito em baixo AND. . ." sendo, desta forma, possível representá-las por árvores. A conclusão é mais claramente diferente em cada caso. As conclusões do Aardvark são decisões sobre animais e podem ser corretas ou erradas, enquanto que o jogo da velha tem como conclusões jogadas que somente se provarão corretas ou erradas posteriormente no jogo. Os programas também aprendem de modo diferente. O programa Aardvark pergunta a um humano como poderá evitar o erro que tiver feito, enquanto que jogo da velha suprime diretamente a regra que levou ao insucesso, do conjunto de todas as regras possíveis, para evitar repetir o erro. A principal diferença entre os dois programas é que o Aardvark não pode saber de antemão todas as regras que podem ser usadas para distinguir os animais, enquanto que o programa do jogo da velha não somente pode, mas, efetivamente, faz uma lista de todas as regras possíveis.

O Jogo da Velha e a Ação de Jogar

Há muito o que aperfeiçoar e estudar ainda no programa do jogo da velha. Em particular, comparando esta maneira de jogar e o enfoque minimax tradicional, é fácil descobrir maneiras de aperfeiçoar este e outros sistemas especialistas. Por exemplo, há um jogo que o programa nunca aprenderá a evitar a derrota. É o mesmo jogo que o programa de uma camada de X ou de O perdeu. De fato, uma análise cuidadosa do programa mostrará que ele somente considera uma jogada à frente, para eliminar movimentos executados por regras ruins. Assim, permanece ainda neste novo programa a preocupação com a árvore de movimentos! Não é difícil modificar este programa, ampliando a regra de seleção ao equivalente a uma busca de duas camadas, sendo necessário, para isto, simplesmente examinar bem o porquê da impossibilidade do programa aprender a não perder aquele jogo em particular, e verificar o que é necessário fazer para que isto não ocorra.

Um Sistema Especialista Universal?

Este capítulo tendeu a enfatizar a utilidade do enfoque de AI que utilize sitemas especialistas, ou programas baseados em regras. É necessário admitir, entretanto, ser difícil acreditar que um jogo tal como xadrez possa ser jogado por um programa baseado em regras, como um sistema de inferências tal como o usado para o jogo da velha. Isto ocorre porque o simples armazenamento, sem pensar na comparação, de todas as posições possíveis do tabuleiro de xadrez é claramente impossível — seu número é grande demais. É possível jogar xadrez com um programa baseado em regras, mas as condições não serão nunca simplesmente as posições do tabuleiro; deve haver também alguma espécie de heurística geral acoplada com um exame local da situação atual da partida. Este raciocínio leva à importante conclusão de que a maior dificuldade em aplicar sistemas baseados em regras a uma ampla gama de problemas de AI consiste usualmente na obtenção de uma boa representação para as condições e as conclusões.

Falta também aos sistemas especialistas a habilidade fundamental que têm os humanos, de inventar e investigar novas regras, ou seja, aquela parte do pensamento que corresponde ao que se denomina criatividade. Isto é verdade e a maioria dos sistemas especialistas tem que obter suas regras de um especialista humano disponível. Entretanto, um ou dois programas experimentais são capazes não somente de processar suas regras para obter outras regras derivadas delas, mas também de deduzir regras completamente novas a partir do exame da informação de entrada. Tente melhorar o programa Aardvark, removendo regras redundantes e especulando sobre a existência de novas regras, sabendo de antemão que esta não é uma tarefa fácil!

5

A estrutura da memória,

Uma grande parte de qualquer computador é usada para o armazenamento dos dados. Entretanto, as memórias dos computadores são muito diferentes das humanas, já que as informações armazenadas nas memórias humanas são acessíveis de maneiras não disponíveis para o acesso às informações que estão nas memórias dos computadores. Atualmente, as memórias dos computadores são usadas mais como modelos de livros do que como modelos do cérebro humano.

Para obter uma peça de informação armazenada em um computador, é necessário saber exatamente onde ela está. Por exemplo, para encontrar o número do telefone de alguém, cujo nome seja dado, um banco de dados computacional poderá ser usado exatamente como um catálogo telefônico. Entretanto, dadas várias informações sobre a pessoa, mas não seu nome, o banco de dados computacional é tão inútil quanto o catálogo telefônico de assinantes. Mesmo que estejam disponíveis fotografias da pessoa e toda a história de sua vida, isto de nada adiantará se seu nome não for conhecido, para a recuperação de seu número telefônico de um banco de dados computacional. Compare isto com as inúmeras maneiras de se obter a mesma informação de um cérebro humano. Uma fotografia, uma descrição ou mesmo um fragmento do nome são muitas vezes suficientes para que o número de telefone da pessoa seja lembrado. A diferença parece provir do fato de que a memória humana está muito mais intimamente ligada ao processamento da informação que as memórias computacionais.

A Natureza da Memória Humana

É claro que a memória humana não é um simples mecanismo! De fato, é mais provável que não seja nem um unico sistema! Os psicólogos têm tradicionalmente identificado dois tipos de memória — de longo prazo e de curto prazo. O entendimento mais recente é o de que a memória não pode ser realmente classificada nestas duas categorias tão convenientes, e consiste, de fato, em uma grande coleção de muitos conjuntos de funções de memória. É também claro que a própria memória não está inteiramente separada dos processos de pensamento e de raciocínio. Em outras palavras, a identificação de um componente de memória no pensamento humano pode ser mais uma conveniência do que corresponder a uma divisão real de funções. Apesar de este poder ser o caso, na prática, há muito a ganhar no estudo da memória separadamente do pensamento.

Um pouco de introspecção confirmará que há uma função de memória associada com cada um dos sentidos; embora não igualmente desenvolvidos, é possível recordar imagens, sons, o tato, o gosto e o cheiro. Além destas memórias associadas aos sentidos, existe ainda um sistema geral de memória que parece estar diretamente relacionado com o raciocínio. Esta memória geral constitui um "modelo do mundo" interno que permite a recordação de objetos, suas propriedades e suas relações uns com os outros. É esta forma mais geral de memória e a maneira que ela interage com a linguagem que é do maior interesse prático à AI.

Esta memória geral é, pela sua própria natureza, uma memória de "longo termo", sendo as memórias de curto termo tipos mais simples de memórias. O melhor exemplo de memória de curto tempo consiste em tentar lembrar um número de telefone que se acabou de ouvir. Em geral, é uma coleção isolada de algarismos que pode ser corretamente lembrada somente por um curto período de tempo. Este tipo de memória pode ser explicado admitindo-se que a seqüência de algarismos é mantida armazenada usando um processo de reciclagem, que evita o problema da armazenagem não deixando que os dados descansem! Por exemplo, nos primeiros computadores, linhas de retardamento de mercúrio eram usadas como memórias de reciclagem (Veja a Figura 5.1). Os dados eram armazenados como um

fluxo de pulsos acústicos que se propagavam pelo mercúrio até serem apanhados por um microfone em uma extremidade, amplificados e reinjetados no tubo por um alto-falante na outra extremidade. Na linha de retardamento de mercúrio, pode-se ver claramente que os dados são armazenados sendo mantidos sempre em movimento.

Na memória humana de curto prazo, pode-se ver o mesmo tipo de processamento quando alguém procura guardar um número de telefone repetindo-o alto algumas vezes, para aumentar suas chances de lembrá-lo quando necessário. O mecanismo físico da memória de curto prazo poderia bem ser baseado numa atividade elétrica do cérebro que reciclasse os dados em ciclos constituídos de neurnôios (veja o Capítulo 8). Um fato que tende a apoiar esta teoria do ciclo para a memória de curto prazo é que ela parece estar limitada ao armazenamento simultâneo de somente sete peças de informação, em média. Esta limitação de tamanho fixo pode corresponder à capacidade física do ciclo.

Na prática, as coisas talvez sejam mais complicadas do que isto, mas, seja lá qual for o mecanismo da memória de curto termo, ele não é tão importante para AI quanto o da memória de longo termo, pois já há bons substitutos para a memória de curto termo — um bloco de notas ou um computador!

Se alguma informação for mantida na memória de curto termo por um lapso suficientemente longo de tempo, será transferida para a memória de longo termo. O mecanismo da memória de longo termo é mais complicado, e não está absolutamente claro se resulta de modificações físicas ou químicas no cérebro. Logo que alguma informação é armazenada na memória de longo termo, ela se liga às outras informações já na memória, e estas ligações formam uma diferença tão essencial entre as memórias de curto e de longo termo como o tempo em que as informações ficam armazenadas! Para demonstrar o número de ligações que uma informação qualquer tem com as outras na memória, tudo o que é preciso é pensar em alguma coisa e, então, anotar tudo o que "vier à mente" em seguida. A memória de longo termo assemelha-se mais a uma rede de informações do que a um bloco de anotações.

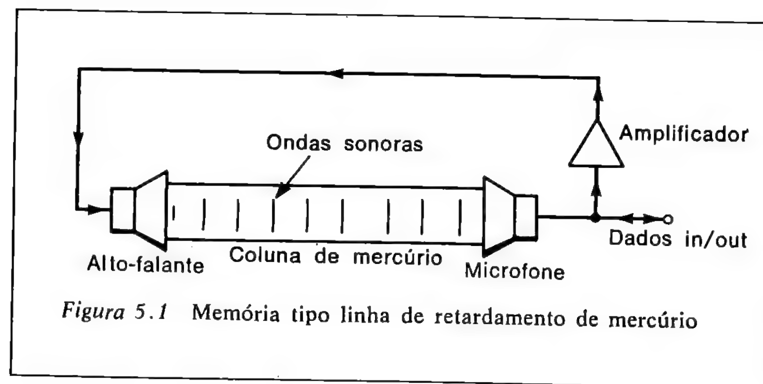


Figura 5.1 Memória tipo linha de retardamento de mercúrio

A Natureza da Memória dos Computadores

A maioria dos computadores separa as funções de memória e de processamento em unidades bastante distintas, a CPU e a RAM. Estas unidades interagem de maneiras muito restritas, se comparadas com o pensamento e a memória humanas. Por exemplo, uma memória de computador armazenará uma informação sem nenhuma modificação e reproduzirá exatamente o que for pedido, e nada mais. Efetivamente, se uma memória computacional trouxesse de volta mais do que foi pedido, ou modificasse os dados armazenados de alguma maneira, seria mandada para conserto! Neste sentido, as memórias computacionais são "passivas" e desempenham um papel tão importante no processamento quanto o lápis e o papel desempenham na aritmética.

As memórias computacionais são ainda mais restritas pelo fato de somente poderem armazenar números de um tamanho fixo. Entretanto, esta restrição advém mais da tecnologia usada para implementar as memórias do que do seu princípio de operação. A característica mais importante da memória computacional é que ela é um armazenamento "endereçável", sendo cada item de informação armazenado relacionado a outro item de informação, chamado seu "endereço". A informação é recuperada da memória fornecendo seu endereço (veja a Fi-

gura 5.2). Neste sentido, uma memória endereçável funciona relacionando pares de itens de informação:

item 1	*item* 2
endereço	dado

Usualmente, a forma do primeiro item de dados, o endereço, é simples e regular. Por exemplo, uma memória de computador é um armazenamento endereçável que usa um conjunto limitado de números como endereços. O uso de endereços númericos

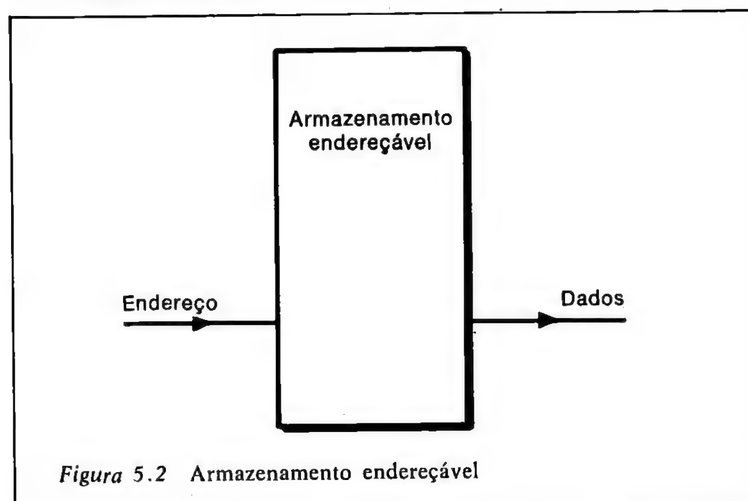


Figura 5.2 Armazenamento endereçável

é tão comum que há uma tendência a pensar que todos os endereços possíveis são numéricos. Entretanto, isto está longe da verdade. Uma utilização típica de um armazenamento endereçável pode ser lembrar uma lista de números de telefones, usando os sobrenomes dos proprietários dos telefones como endereços, sendo que tal armazenamento poderia ser implementado em muitas tecnologias diferentes; por exemplo, tecnologia de papel daria a lista telefônica usual!

Para resumir: o princípio que embasa as memórias computacionais e outros tipos de armazenamento endereçável é o do relacionamento de pares de peças de informação, o endereço e o dado. O endereço é associado com o dado à medida que os dados forem sendo armazenados, e é usado para recuperar o dado da memória. Para assegurar que cada item de dados esteja perfeitamene identificado, cada endereço deve ser único.

O Problema da Recuperação: Memória Associativa

Para alguém familiar com a maneira de operar de uma memória de computador, a descrição feita de um armazenamento endereçável não trouxe nada de novo, com a possível exceção da extensão dos endereços para incluírem quaisquer tipos de dados. O fato de que as memórias computacionais funcionam, são úteis e constituem claramente um tipo de armazenamento associativo, tende a esconder o problema inerente a todos os tipos de memórias associativas. Entretanto, alguém ainda não familiarizado com as memórias computacionais poderia já ter detetado a natureza do problema na semelhança da descrição dos dados e dos endereços. A dificuldade é que, para recuperar qualquer informação armazenada na memória endereçável, é necessário saber seu endereço e, sendo tanto o dado como o endereço informações que podem ser de mesmo tipo, a troco de que é mais provável que se possua o endereço do que o dado? Na prática, o único caso em que uma memória endereçável é de alguma utilidade é quando os endereços são simples, ou formam um sistema simples. Por exemplo, a única razão da utilidade dos catálogos telefônicos é que é mais fácil lembrar nomes do que números de telefone. Em geral, o endereço é uma informação "menor" que pode ser usada como um "guia" que leve a uma quantidade muito maior de informação. Por exemplo, em um Quem é Quem, o nome de uma pessoa é usado como um endereço para as palavras, possivelmente mais que uma centena, que a descrevem.

O uso de um endereço para recuperar dados é simultaneamente a força e a fraqueza dos armazenamentos endereçáveis. Nas aplicações para as quais o uso de um endereço simples é suficiente, não há nada mais eficiente do que um armazenamento endereçável, mas há muitas aplicações importantes que não se enquadram neste esquema simples. Por exemplo, considere a tarefa aparentemente simples de encontrar o nome correspondente a um número de telefone. Se a informação tiver sido armazenada utilizando os nomes como endereços e os números de telefones como os dados, o problema é muito difícil de resolver! Usando um sistema convencional de armazenamento endereçável, a única solução é percorrer todos os dados armazenados até encontrar o número de telefone desejado. A difi-

culdade, neste caso, consiste em que um armazenamento endereçável torna fácil a obtenção dos dados, sendo conhecidos os endereços correspondentes, enquanto que o problema pede o endereço correspondente a um dado! Pode-se pensar que a solução, no caso, consiste em simplesmente usar o número telefônico como o endereço e o nome como dado, mas a dificuldade para se fazer isto é que pode haver mais de um nome por número de telefone, o que violaria o requisito de haver um único endereço para cada item de dados. A dificuldade real com um armazenamento endereçável é que o endereço determina diretamente o dado correspondente, mas o contrário não acontece. (veja a Figura 5.3). Até um certo ponto, este raciocínio perde a semelhança essencial entre os endereços e os dados. Poder-se-ia dizer que a única coisa que distingue o endereço é fato de que ele é conhecido e que o que é desejado é o dado a ele associado. Memórias que podem recuperar endereços correspondentes a itens de dados existem e são chamadas de memórias associativas, ou ainda Memórias Endereçáveis pelo Conteúdo (CAM, do inglês Content Addressable Memory) (veja a Figura 5.4). Observe que uma memória associativa recuperará todos os endereços correspondentes a um item de dados. As memórias associativas são muito úteis e, como podem ser facilmente implementadas em hardware, oferecerá a possibilidade da construção de computadores mais rápidos. No futuro, pode-se esperar encontrar máquinas com os dois tipos de memórias, endereçável e associativa, mas nenhum destes dois tipos de memórias ataca realmente as dificuldades, que se concentram na necessidade de relacionar pares de informações, o endereço e o dado.

Armazenamento Relacional

Uma memória não tem somente que armazenar informações, mas tem também que formar ligações ou relações entre diferentes itens de informação. A sofisticação da memória depende não somente de sua capacidade de armazenamento, mas também da variedade de relacionamento entre as informações que pode manter. O tipo mais simples de memória já considerado, memória de computador, forma ligações entre os pares de itens tais como mostra a Figura 5.5. A noção de um item de informação ser o endereço e outro o dado é inteiramente devida às regras que governam a maneira de estabelecer as ligações. A memória mais sofisticada considerada, a humana, é caracterizada pela ampla gama de ligações feitas entre os itens de informação armazenados. De fato, tais ligações são feitas de maneira tão livre, que as idéias de endereço e de dado têm que ser modificadas. Um item de informação pode ser usado para lembrar outros, que, por sua vez, pode levar a lembrar outro, e assim por diante. Um item de informação será um endereço se for conhecido e usado para lembrar outros itens a ele associados.

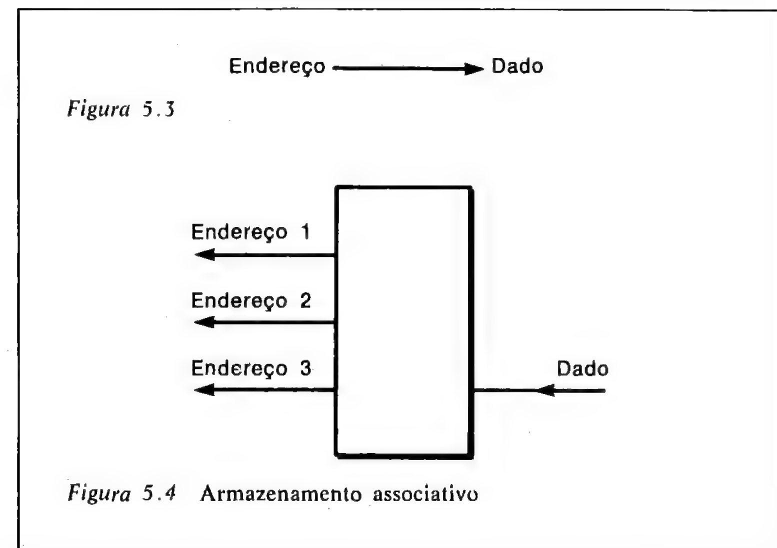


Figura 5.3

Figura 5.4 Armazenamento associativo

Isto sugere que, para construir memórias mais sofisticadas, vale a pena investigar métodos de aumentar as maneiras de estabelecer ligações entre os itens de informação. O aperfeiçoamento mais óbvio consiste em permitir que exista um número qualquer de ligações em quaisquer direções, ligando quaisquer pares de itens. Tal esquema leva a um armazenamento relacional. Por exemplo, na Figura 5.6 itens separados de informação são ligados de forma tal que, a partir do "endereço", pode ser encontrado o "nome" ou o "número do telefone", e vice-versa. Observe que um armazenamento relacional pode

dar resultados inesperados. Por exemplo (usando a estrutura da Figura 5.6), usando "nome" como um endereço, para determinar a "idade", o resultado é somente uma informação, ou seja, a idade da pessoa. Ao contrário, usando "idade" como endereço, o resultado será uma lista de todas as pessoas daquela idade. A memória relacional não faz nenhuma distinção apreciável entre endereços e dados, mas não se pode esperar que todas suas ligações funcionem da mesma maneira.

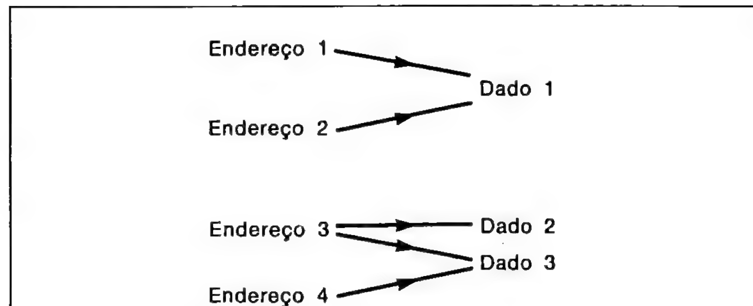


Figura 5.5 Relacionamento entre os endereços e os dados

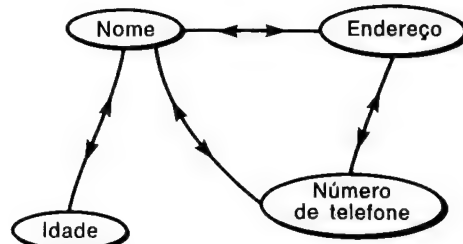


Figura 5.6 Armazenamento relacional

Uma forma modificada e muito prática do armazenamento relacional está na base do amplo espectro de *softwares* comerciais usualmente denominados "bancos de dados relacionais". Um banco de dados relacional armazena listas de informações e usa "apontadores" (ou ponteiros) para permitir um armazenamento econômico das informações, que poderão ser recuperadas usando uma grande variedade de endereços. Os bancos de dados relacionais têm grande potencial de tornarem-se parte importante da pesquisa de AI nos próximos anos; entretanto, o banco de dado relacional não representa a última palavra em sofisticação de memória.

Armazenamentos Conceituais

As ligações usadas para associar itens até agora têm sido todas do mesmo tipo, setas ou apontamentos anônimos, que simplesmente ligam os itens de informações, sem indicar o que causa esta associação. Uma memória humana é muito diferente, pois liga os itens com relações possuindo nomes. Por exemplo, na Figura 5.7,* as relações entre os itens de informação TOM, CAT, TAIL, FUR, SOFT e MALE são mostradas, podendo-se

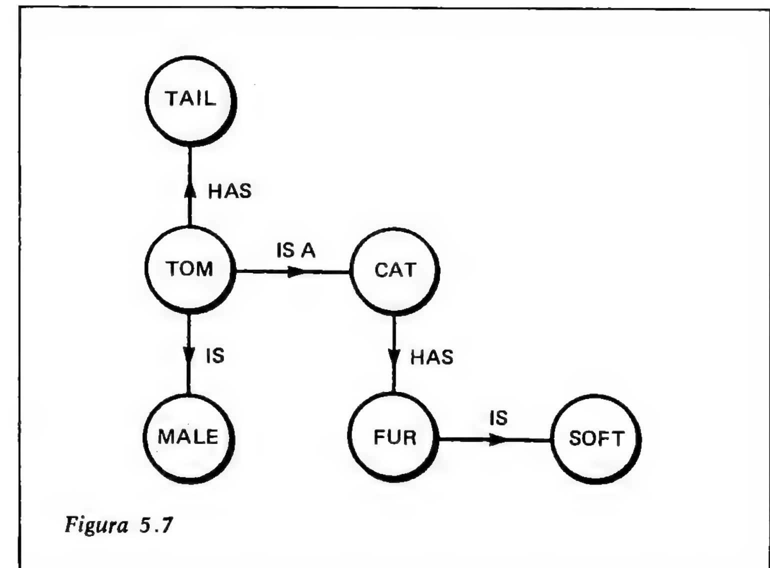


Figura 5.7

* N.T.: Neste exemplo, serão mantidas as palavras e frases em inglês, para conservar a lógica do programa apresentado. Sera dada, entretanto a tradução de tudo o que ficar em inglês, na primeira vez que aparecer no texto. Na Figura 5.7, TAIL *é rabo*, HAS é *tem*, TOM é o nome do gato, IS A é *é um*, CAT é *gato*, IS é *é*, MALLE é *macho*, FUR *pêlo* e SOFT é *macio*.

ver que a relação entre TOM e CAT é diferente daquela entre TOM e TAIL, ou seja TOM *IS A* CAT, mas TOM *HAS* TAIL. O fato de que as relações *IS A* e *HAS* são diferentes aparecerá realmente quando a memória estiver sendo pesquisada. Por exemplo, colocada a questão DOES TOM HAVE FUR? (*Tom tem pêlo?*) será necessário encontrar as ocorrências do item TOM e suas relações tipo *HAS*, e também examinar os itens conectados a TOM pela relação *IS A*, que poderão continuar com uma relação tipo *HAS*. Assim, a resposta a esta questão será encontrada examinando-se a relação *HAS* saindo de *CAT*, ou seja:

TOM—IS A—CAT—HAS—FUR

indica que TOM efetivamente em pêlo.

As relações com nomes entre itens de informação podem ser denominadas "conceitos", resultando em que as memórias deste tipo são denominadas "armazenamento conceitual". Estas podem, em geral, ser muito complicadas, mas é possível produzir um pequeno exemplo em BASIC para mostrar como as coisas funcionam e fornecer um ponto inicial para quem quiser experimentar com este tipo de memória.

Um Programa Geral de Banco de Dados Conceitual

Para escrever um programa que armazene um sistema de itens com relações tendo nomes, é necessário, inicialmente, resolver o problema da representação destas ligações. Cada item de informação pode ser armazenado em um elemento de um *array* alfanumérico, seja W$. Quando um item for encontrado, será simplesmente armazenado na próxima posição livre deste *array*. Um item ligado a este estará, obviamente, também armazenado no *array* W$ e é possível indicar qual é, armazenando seu índice em outro *array* P. Por exemplo, se o *array* W$ contiver

```
W$(1) = "TOM"
W$(2) = "TAIL"
W$(3) = "CAT"
W$(4) = "FUR"
```


a ligação entre TOM e TAIL e CAT e FUR seria armazenada em P na forma

```
P(1) = 2
P(2) = 0
P(3) = 4
P(4) = 0
```

Em outras palavras, o item ao qual está ligado W$(1) tem seu índice armazenado em P(1). Um valor zero em P(I) indica que o item não está ligado a nada. Este uso combinado dos *arrays* W$ e P resulta em um baco de dados relacional simples, já que, embora as ligações estejam todas representadas, ainda são anônimas. A mais simples maneira de armazenar os nomes das ligações é como itens de informação em W$! Para associar o nome de ligação com seu uso, será necessário ainda outro *array* R para armazenar o índice do nome. Assim:

W$(I)	é um item
P(I)	é o índice do item ao qual ele está ligado, ou seja, está ligado ao item W$(P(I))
R(I)	é o índice do nome da relação, ou seja, W$(R(I)) é o nome da relação

Por exemplo, a inclùsão dos nomes das relações ao conteúdo anterior de W$ dá

```
W$(1) = "TOM":P(1) = 2:R(1) = 5
W$(2) = "TAIL": P(2) = 2:R(2) = 0
W$(3) = "CAT": P(3) = 2:R(3) = 6
W$(4) = "FUR": P(4) = 2:R(4) = 0
W$(5) = "HAS": P(5) = 2:R(5) = 0
W$(6) = "IS A": P(6) = 2:R(6) = 0
```

Assim, W$(1), TOM, está ligado a W$(P(1)), TAIL, e a relação é W$ (R(1)), *HAS*. Observe que o valor de R(1) é feito igual a zero quando não houver ligação à etiqueta.

O único problema remanescente é o que acontecerá quando houver mais do que uma ligação a partir de um item. Isto poderia ser resolvido transformando-se os *arrays* P e R em bidimensionais e usando-os para armazenar todas as ligações e seus respectivos nomes para cada item. Assim, P(1,1) armazenaria a primeira ligação do primeiro item e P(1,2) sua segunda ligação, e assim por diante. A solução adotada pelo pro-

grama adiante é mais simples, mas potencialmente mais lenta. Cada vez que um item aparece, é armazenado em W$, mesmo que já esteja aí presente. Desta maneira, para adicionar TOM IS MALE à memória, basta armazenar os três itens, TOM, IS e MALE em W$, ajustando-se P e R correspondentemente, resultando:

```
W$(7) = "TOM":P(7) = 8:R(7) = 9
W$(8) = "MALE":P(8) = 0:R(8) = 0
W$(9) = "IS A": P(9) = 0:R(9) = 0
```

Na prática, esta duplicação de itens permite a inclusão de nova informação na memória sem a necessidade de verificar se esta já não está incluída, e não causa nenhum problema prático.

Estando resolvido o problema de como representar a estrutura de um armazenamento conceitual, basta decidir como introduzir e recuperar informação nesta estrutura. A maneira mais fácil de permitir ao usuário armazenar informação é através da frase

x relação *y*

onde *x* e *y* são itens a ser armazenados, ligados pela ligação chamada "relação", tal como por exemplo

```
TOM IS A CAT
TOM IS MALE
TOM HAS TAIL
CAT HAS FUR
FUR IS SOFT
```

descrevem a informação mostrada na Figura 5.7. O uso desta forma limitada de frase garante ao programa a possibilidade de separar *x* e *y* da relação. A regra usada é que a primeira palavra é o item *x*, a última é o item *y*, e quaisquer palavras intermediárias descrevem a relação.

Logo que as informações estejam armazenadas na memória, é preciso ter uma maneira simples de responder a questões quanto ao seu conteúdo. Usando a forma de representação descrita, é possível implementar muitos tipos de rotinas para responder a perguntas, mas, para manter o programa simples, este somente tentará responder aos seguintes tipos de pergunta:

TELL ME ABOUT *x*	*(diga algo sobre x)*
DOES *x* HAVE *y*	*(x tem y?)*
IS *x* A *y*	*(é x um y?)*

Mais uma vez, devido à forma restrita destas perguntas, é fácil para o programa detectar o tipo de questão e os itens citados nela. A primeira questão é detectada pela primeira palavra ser TELL e a última é tomada por *x*; semelhantemente, a segunda e a terceira questões são detectadas pela primeira palavra ser DOES e IS, respectivamente, sendo a segunda palavra tomada por *x* e a última por *y*.

O que acontece quando cada uma destas questões é detectada, varia com a questão. Para responder a uma questão TELL ME ABOUT *x*, o programa deverá procurar todas as ocorrências de *x* em W$ e imprimir a relação W$(R(I)) e o item ao qual está relacionada (W$(P(I)). Entretanto, se o programa fizesse somente isto, a resposta TELL ME ABOUT TOM seria TOM IS A CAT — HAS TAIL — IS MALE, mas, seguindo a relação tipo IS A para CAT poder-se-ia também descobrir que TOM HAS FUR — IS SOFT. Assim, para responder à questão em sua totalidade, o programa deve seguir todas as ligações que saem de TOM através de toda a memória. Assim, a resposta a uma questão do tipo TELL ME ABOUT *x* conterá todas as relações que *x* tiver com todo o restante da memória! Para responder a uma questão do tipo DOES *x* HAVE *y* será necessário não somente examinar as relações tipo HAS diretamente ligadas a *x*, mas também as relações tipo HAS ligadas indiretamente a *x* via outras relações tipo IS A. Assim, DOES TOM HAVE FUR seria respondida pela cadeia seguinte de relações:

TOM IS A CAT HAS FUR

Da mesma forma, a questão IS *x* A *y* deve ser respondida seguindo-se as ligações tipo IS A. Assim, se CAT IS AN ANIMAL for adicionado à memória, a resposta à questão IS TOM

AN ANIMAL seria respondida seguindo-se a cadeia seguinte de relações:

TOM IS A CAT IS AN ANIMAL

A necessidade de examinar não somente as relações diretas, mas também seguir as cadeias de relações de um determinado tipo é usualmente difícil de implementar sem o recurso a técnicas avançadas, especificamente à recursão. A maneira mais simples de implementar tais cadeias em BASIC consiste em usar uma pilha para armazenar os resultados intermediários. Uma pilha é simplesmente uma memória temporária que é usada via duas operações: PUSH (*empurre*) e PULL (*puxe*), sendo que a primeira armazena um item na pilha e a segunda recupera um item. A ordem em que os itens são recuperados é oposta à em que são armazenados, ou seja, o último armazenado será o primeiro recuperado. Para esta aplicação, a ordem em que os itens são recuperados não é importante, sendo a pilha simplesmente usada como armazenamento temporário para as ligações relevantes que ainda não tiverem sido examinadas.

É mais fácil compreender a maneira de funcionar desta lógica através de um exemplo de como é tratada a questão DOES *x* HAVE *y* aplicada aos dados da Figura 5.7. A questão DOES TOM HAVE FUR é respondida pesquisando-se inicialmente W$ em busca da palavra TOM e, quando esta for achada, o tipo de relacionamento será verificado. Se for um relacionamento tipo HAS *y* será necessário testar *y* para ver se é FUR e, se for, estará encontrada a resposta. Se não for, a pesquisa continuará em busca de outra ocorrência de TOM em W$. Entretanto, se for encontrada uma relação tipo IS A *y*, o índice de *y* deverá ser armazenado na pilha e, quando todo o *array* W$ tiver sido pesquisado, será necessário repetir a busca para todos os itens que tiverem sido armazenados na pilha, até esvaziá-la.

No caso das informações da Figura 5.7, a primeira ocorrência de TOM tem um relacionamento tipo HAS, mas não responde à questão. A ocorrência de TOM IS não pode fornecer nenhuma informação de interesse no caso, sendo, portanto, ignorada, mas a ocorrência de TOM IS A resultará no índice de CAT ser armazenado na pilha. Ao final da primeira varredura

por W$, a pergunta não estará ainda respondida, mas há uma informação na pilha que deve ser investigada. Recuperada esta informação da pilha, o *array* W$ será pesquisado em busca das ocorrências de CAT HAS e CAT IS A e, logo na primeira ocorrência de CAT HAS a pergunta é respondida e o processamento pode ser terminado. Em princípio, entretanto, quaisquer relações tipo IS A encontradas nesta nova busca deveriam ser armazenadas na pilha e, caso a resposta ainda não tivesse sido encontrada, seria necessária uma nova varredura de W$. Assim, a pilha é usada para armazenar temporariamente os itens que deverão posteriormente ser pesquisados em busca de relações tipo HAS que respondem à questão, ou em busca de relações tipo IS A, que estenderão novamente a busca.

Juntando tudo isto, resulta o programa*

```
  10 GOSUB 1000
  20 INPUT A$
  30 I$=A$
  40 GOSUB 2000
  50 IF K$="TELL" THEN GOSUB 3000:GOTO 20
  60 IF K$="DOES" THEN GOSUB 4000:GOTO 20
  70 IF K$="IS" THEN GOSUB 5000:GOTO 20
  80 X$=K$
  90 GOSUB 6000
 100 GOSUB 7000
 110 GOTO 20

1000 N=50
1010 DIM W$(N)
1020 DIM T(N)
1030 DIM R(N)
1040 DIM P(N)
1050 R=0
1060 DIM M(N)
1070 T=1
1080 RETURN
```

* N.T.: As traduções das mensagens de saída do programa são:
na linha 3200 — I DON'T KNOW ANYTHING ABOUT é *não sei nada sobre*
na linha 3210 — PERHAPS YOU WOULD LIKE TO TELL ME ABOUT é *talvez você deseje dizer-me algo sobre*
na linha 4170 — NO INFORMATION ON é *não há informações sobre*
na linha 4180 — NOT AS FAR AS I KNOW é *não que eu saiba*

```
2000 K$=""
2010 IF LEN(T$)=0 THEN RETURN
2015 IF LEFT$(T$,1)<>" " THEN GOTO 2040
2020 T$=RIGHT$(T$,LEN(T$)-1)
2030 GOTO 2010
2040 K$=K$+LEFT$(T$,1)
2050 T$=RIGHT$(T$,LEN(T$)-1)
2060 IF LEFT$(T$,1)=" " THEN GOTO 2080
2070 GOTO 2040
2080 IF LEN(T$)=0 THEN RETURN
2090 T$=RIGHT$(T$,LEN(T$)-1)
2100 IF LEFT$(T$,1)<>" " THEN RETURN
2110 GOTO 2080

3000 X$=""
3010 GOSUB 2000
3020 IF LEN(T$)=0 THEN GOTO 3040
3030 GOTO 3010
3040 X$=K$
3045 F=0
3050 FOR I=1 TO S
3060 IF W$(I)<>X$ THEN GOTO 3110
3064 IF P(I)=0 THEN GOTO 3110
3065 F=F+1
3070 IF F=1 THEN PRINT X$;" ";
3080 PRINT W$(R(I));" ";
3090 PRINT W$(P(I))
3095 D=P(I)
3100 GOSUB 8000
3110 NEXT I
3120 IF F=0 THEN GOTO 3200
3130 IF T=1 THEN RETURN
3140 GOSUB 8100
3150 X$=W$(D)
3160 GOTO 3050
3200 PRINT "I DON'T KNOW ANYTHING ABOUT ";X$
3210 PRINT "PERHAPS YOU WOULD LIKE TO TELL
     ME ABOUT ";X$
3220 RETURN

4000 GOSUB 2000
4010 X$=K$
4020 IF LEN(T$)=0 THEN GOTO 4050
4030 GOSUB 2000
4040 GOTO 4020
4050 Y$=K$
4055 K$=X$
4060 F=0:A=0
4070 FOR I=1 TO S
4080 IF W$(I)<>X$ THEN GOTO 4110
4085 F=1
4090 IF W$(R(I))=" IS A" THEN D=P(I):GOSUB 8000
4100 IF W$(R(I))=" HAS A" THEN D=P(I):GOSUB 8000:
     GOSUB 8200
4105 IF W$(R(I))=" HAS" THEN D(P(I):GOSUB 8000:
     GOSUB 8200
4110 NEXT I
4120 IF T=1 THEN GOTO  4170
4130 GOSUB 8100
4140 IF D=0 THEN GOTO 4120
4150 X$=W$(D)
4160 GOTO 4070
4170 IF F=0 THEN PRINT "NO INFORMATION ON ";K$
4180 IF F=1 AND A=0 THEN PRINT "NOT AS FAR AS
     I KNOW"
4190 IF F=1 AND A=1 THEN PRINT "YES"
4200 RETURN

5000 GOSUB 2000
5005 X$=K$
5010 GOSUB 2000
5020 IF LEN(T$)<>0 THEN GOTO 5010
5030 Y$=K$
5040 K$=X$
5050 F=0:A=0
5060 FOR I=1 TO S
5070 IF W$(I)<>X$ THEN GOTO 5100
5080 F=1
5090 IF W$(R(I))=" IS A" THEN D=P(I):GOSUB 8000:
     GOSUB 8200
5100 NEXT I
5110 IF T=1 THEN GOTO 4170
5120 GOSUB 8100
5130 IF D<>0 THEN X$=W$(D):GOTO 5060
5140 GOTO 5110

6000 R$=""
6010 GOSUB 2000
6020 IF LEN(T$)=0 THEN GOTO 6050
6030 R$=R$+" "+K$
6040 GOTO 6010
6050 Y$=K$
6060 RETURN

7000 S=S+1
7010 W$(S)=X$
7020 T(S)=4
7030 W$(S+1)=R$
7040 T(S+1)=2
7050 W$(S+2)=Y$
7060 R(S)=S+1
7070 P(S)=S+2
```

```
7080 S=S+2
7090 RETURN

8000 IF T>N THEN RETURN
8010 M(T)=D
8020 T=T+1
8030 RETURN

8100 IF T=1 THEN RETURN
8110 T=T-1
8120 D=M(T)
8130 RETURN

8200 IF W$(P(I))=Y$ THEN A=1
8210 RETURN
```

Para auxiliar a compreensão do programa, sua estrutura é:

10	programa principal recebe os dados e decide se é uma pergunta ou novos dados
1000	início
2000	remove a próxima palavra em T$ e guarda em K$
3000	sub-rotina TELL
4000	sub-rotina DOES
5000	sub-rotina IS
6000	acrescenta informações em W$
8000	armazena número em D na pilha
8100	retira número da pilha e guarda em D
8200	compara W$(P(I) com Y$ e faz A = 1 se forem iguais

Usando o Banco de Dados Conceitual

O programa dado implementa somente algumas poucas das idéias de um banco de dados conceitual e, assim mesmo, já se comporta de maneira muito interessante, como mostra sua listagem de saída dada em seguida. Para maior clareza, os dados submetidos ao programa são mostrados em letras maiúsculas e suas respostas em minúsculas.

Há muitos outros aperfeiçoamentos que podem ser introduzidos neste programa. No nível mais simples de implementação, o método usado para armazenar os nomes das relações é muito ineficiente. Também, o programa deveria periodicamente varrer


```
TOM HAS A TAIL
TOM IS MALE
TOM IS A CAT
CAT HAS FUR
FUR IS SOFT
IS TOM A CAT
yes
DOES TOM HAVE A TAIL
yes
TAIL HAS STRIPES
CAT IS A ANIMAL
IS TOM A ANIMAL
yes
DOES TOM HAVE STRIPES
yes
DOES CAT HAVE STRIPES
not as far as I know
```

o *array* W$, retirando as ocorrências múltiplas do nome de alguma relação e ajustando os apontadores de relacionamento, em R(I), para apontarem todos para a ocorrência única de cada relação. A um nível mais sofisticado, seria possível reconhecer diferentes categorias ou classes de objetos na memória. Por exemplo TOM é um único objeto, mas CAT é uma classe à qual podem pertencer outros objetos. Tais classes podem ser detectadas, pois são sempre seguidas por uma relação tipo IS A. De fato, o programa já inclui o *array* T, que registra o tipo de cada item, à medida que é introduzido, embora esta informação acabe não sendo usada pelo programa.

T(I) = 1 indica que o item é um objeto
T(I) = 2 indica que o item é uma relação
T(I) = 3 indica que o item é uma classe
T(I) = 4 indica que o item é uma classe ou um objeto

É possível, também, incluir novas relações no programa e processá-las em resposta a perguntas. Por exemplo, TOM IS A CAT implica que TOM tem todos os itens que CAT possui,

mas que o contrário não é verdade. De fato, uma nova relação interessante é:

CAT EXAMPLE IS TOM (*exemplo de* CAT *é* TOM)

Dada esta nova relação, quaisquer itens que TOM HAS poderiam ser também itens que CAT HAS e, desta maneira, seria possível fazer a resposta a DOES CAT HAVE STRIPES (CAT *tem listras?*)* ser AT LEAST SOMETIMES (*pelo menos algumas vezes*). Se aparecessem muitas outras ocorrências de gatos com listras, o programa poderia até concluir pela resposta CAT HAS STRIPES.

Executando o programa dado, pode-se descobrir muitas maneiras de examinar e reorganizar as relações entre os itens. Há um perigo que deve ser considerado ao utilizar o programa, o da ocorrência de relações circulares. Um relacionamento que inclua tanto TOM IS A CAT como CAT IS A TOM resulta em um ciclo infinito no programa! Obviamente, o perigo não consiste na inclusão de tais definições tão ostensivamente circulares, mas que elas sejam criadas por uma longa cadeia de relações tipo IS A.

Lembrando-se de Pensar

A maneira pela qual a memória conceitual é envolvida no pensamento, além de usada como armazenamento, fica óbvia na discussão anterior. O tipo de relação existente entre dois itens governa o tipo de raciocínio que poderá ser aplicado. O mais interessante sobre o programa dado neste capítulo é que ele armazena o tipo de relacionamento exatamente da mesma maneira que os outros itens, o que significa que, pelo menos em princípio, seria possível estabelecer relações entre relações! A conexão entre a memória e o raciocínio ainda precisa ser muito estudada para ter explicações razoáveis, o que sugere que há muito trabalho a fazer na busca de meios de combinar memória conceitual com sistemas especialistas.

---

* N.T.: A única palavra ainda não traduzida é "stripes", que quer dizer listras.

6

Reconhecimento de padrões

De certa forma, é difícil ver temas gerais em AI. Algumas idéias, como as árvores, aparecem em muitas aplicações, mas, freqüentemente, é difícil ver por quê. Isto reflete o fato de que, até o momento, não existe nenhuma teoria geral de AI. Há, entretanto, algumas áreas de AI que já produziram *suas* teorias gerais, que alguns defendem como teoria geral da AI. "Reconhecimento de Padrões" é uma destas áreas.

Embora o interesse original do estudo de reconhecimento de padrões tenha sido na área de visão artificial, resultando em que padrão tivesse o significado de padrão visual, o termo pode ser facilmente estendido de maneira a incluir padrões sonoros, ou de eventos de quaisquer tipos. Por exemplo, o problema dos jogos de duas pessoas, tais como o jogo da velha ou xadrez, pode ser considerado em termos da determinação de padrões de movimentos e peças nos tabuleiros. É deste ponto de vista que se pode ver como o reconhecimento de padrões pode dar uma visão total da área de AI. Na realidade, a situação não é tão clara, já que a maioria das áreas de AI usam técnicas de reconhecimento de padrões, mas sempre conjuntamente com outras técnicas e teorias que são igualmente importantes. Se, entretanto, alguém achar que um assunto denominado "reconhecimento de padrões" é excessivamente especializado para ser de seu interesse, estará completamente enganado! A atividade relacionada com o reconhecimento ou a detecção de um "padrão" constitui um aspecto muito geral do comportamento inteligente para ser ignorada. Em termos das regras IF "condição" THEN "conclusão" que foram descritas no Capítulo 4, pode-se

dizer que reconhecimento de padrões trata do complicado problema da implementação da parte da regra chamada de "condição".

Reconhecimento e Aprendizado

Os melhores "reconhecedores" de uso geral que podem ser estudados são exatamente os humanos, que, embora possam ser superados em tarefas específicas, até o momento não conseguem ser superados em versatilidade. De fato, conforme mencionado em outra parte deste livro, os humanos são quase excessivamente bons em reconhecimento de padrões, chegando a identificá-los mesmo onde não existem. A maioria das maneiras de atacar o reconhecimento de padrões artificialmente são tentativas de copiar a maneira humana de fazê-lo. Há, entretanto, duas maneiras principais de atacar o problema. Pode-se tanto ignorar o fato de que os humanos *aprendem* a reconhecer as coisas, como tornar este fato fundamental para o método. Um humano recém-nascido reconhece poucas coisas, mas, à medida que é exposto repetidamente a alguns padrões, passa rapidamente a reconhecer muitas coisas. Mesmo após muitas outras habilidades mentais terem sido plenamente desenvolvidas, o humano maduro continua aprendendo a reconhecer novos padrões. Um humano bem-dotado e aplicado poderia mesmo reconhecer as palavras de uma língua estrangeira, ou mesmo um novo alfabeto escrito. De fato, todos esperam reconhecer uma nova face após tê-la visto somente uma ou duas vezes e ficam muito aborrecidos se esta sua habilidade, ou a do outro, deixar de funcionar! O fato de que os humanos aprendem a reconhecer padrões levou a um enfoque geral do reconhecimento artificial de padrões, que envolve o "treinamento" de um programa, ou de uma máquina, para reconhecer padrões. Isto levou também a que grande parte do estudo do aprendizado tenha ficado imerso no reconhecimento de padrões. O outro enfoque consiste em considerar o aprendizado como uma parte incidental do reconhecimento de padrões, focalizando o interesse no "reconhecedor" e não em como ele se tornou um.

Para aplicações práticas, freqüentemente parece melhor tentar construir uma máquina, ou escrever um programa, que reconheça algo de interesse do que atacar o problema geral de desenvolver uma máquina que aprenda a reconhecer uma gama de coisas. Por exemplo, para reconhecer as letras do alfabeto automaticamente (um problema de grande interesse para os Correios!) por que não escrever um programa que reconheça a letra A, depois a letra B etc...? A dificuldade consiste em que é necessário *saber* como reconhecer a letra A para ser possível escrever um programa para fazê-lo e, embora todos reconheçam facilmente a letra A quando a vêm, muitos têm provavelmente uma idéia muito vaga de como isto é feito. Adotando o caminho aparentemente mais difícil e tentando escrever um programa ou construir uma máquina que aprenda a reconhecer o alfabeto, não será necessário saber como um humano atinge este resultado. Tudo o que será necessário fazer é apresentar exemplos das letras para a máquina e deixá-la classificá-los. Se ela classificar um exemplo corretamente, tudo estará bem, se, ao contrário, ela errar uma classificação, tudo o que o usuário humano terá de fazer será informá-la do erro e da classificação correta. Em outras palavras, ao invés de embutir no programa o conhecimento de como os humanos reconhecem as letras, pode-se deixar ao programa a tarefa de aprender através de exemplos, treinando-o a reconhecer qualquer coisa que os humanos reconheçam.

Conforme já mencionado no Capítulo 1, estas duas maneiras de atacar problemas — o enfoque aprendizado/desenvolvimento e o enfoque da solução direta — podem ser encontrados em todas as áreas de AI, dando origem a muitas divisões entre os entusiastas do assunto e muitas discussões são dedicadas a decidir qual é o melhor. Alguns interessados em AI acreditam que não há necessidade de um conhecimento detalhado da inteligência e do comportamento humanos, pois tudo o que é necessário fazer é construir a máquina ideal de aprendizado, ligá-la e deixá-la aprender! Já outros são de opinião de que isto vai demorar muito e, de qualquer maneira, para construir tal máquina será necessário conhecimento de como os humanos pensam. É impossível decidir quem está mais certo, pois ambos os pontos de vista têm certa dose de razão. Atualmente, a maneira que parece mais prática para resolver estes problemas talvez seja construir e usar alguma coisa que funcione, mesmo que não se saiba exatamente como funciona.

Afinal, este é o enfoque inerente a todo o uso da heurística, introduzida nos Capítulos 2 e 3.

Embora programas e máquinas que aprendam sejam importante parte do reconhecimento de padrões, o aprendizado é um assunto muito interessante por si mesmo. Em seguida, vale a pena examinar um exemplo do método de reconhecimento "direto" e, depois, considerar o enfoque usando-se o aprendizado.

Reconhecendo Imagens

Conforme foi dito no início, o reconhecimento de padrões começou do interesse em processar padrões visuais. Para fazer algum progresso neste tipo de trabalho, é necessária uma câmara de TV e um conversor A-D, ligado a um computador bastante rápido, o que tira o reconhecimento de padrões do alcance da maioria dos proprietários de computadores pessoais. Entretanto, para alguém realmente interessado em trabalhar no assunto, é bom saber que é possível construir um digitador de imagens por cerca de US$700.00, além de que, usando uns poucos diodos sensíveis à luz em um arranjo plano, juntamente com algumas poucas lentes, é possível construir um dispositivo de entrada visual de baixa resolução por algumas dezenas de dólares. Há dois tipos básicos de imagens visuais que se pode tentar reconhecer: dois tons (por exemplo, branco puro e preto puro) e cinzas, que podem ter todos os tons entre o branco e o preto. Pode-se armazenar uma imagem cinza usando-se um *array* bidimensionais para guardar valores que correspondam ao brilho de cada ponto. O reconhecimento de um padrão em uma imagem cinza, tal como uma fotografia, é um problema excessivamente complicado do ponto de vista do *hardware*, mas, impondo a restrição de que as imagens deverão ser de dois tons, por exemplo, branco e preto, não somente os problemas ficam um pouco mais fáceis de resolver, mas é possível, inclusive, fazer algum reconhecimento de padrões sem um dispositivo visual de entrada. Esta restrição ao branco e preto pode parecer exagerada, mas é suficiente para tratar de um dos problemas mais interessantes e de importância prática de reconhecimento de padrões: o reconhecimento de caracteres.

Uma imagem em branco e preto pode ser armazenada em um computador como um *array* de zeros e uns: um representando preto e um zero representando branco, por exemplo. Isto é muito semelhante ao modo no qual computadores tais como o BBC Micro e o Spectrum especificam seus caracteres definidos pelo usuário. Usando tal método, é possível digitar imagens em branco e preto, desde que o número de pontos brancos e pretos (usualmente chamados de "pixels", abreviatura de "picture elements", *elementos de figura* em inglês) seja pequeno. Para reconhecimento de caracteres, isto não constitui uma grande limitação, pois é suficiente uma grade 8x8 para representar um caráter simples.

Resumindo: as imagens visuais podem ser representadas por uma coleção de valores em *array* bidimensional, cada valor representando o brilho de um ponto da imagem. Para fins de simplicidade, é útil restringir o tratamento a imagens com somente dois valores de brilho — preto e branco — sendo estas representadas por *arrays* contendo somente zeros e uns.

Reconhecendo Características

Considere a tarefa de reconhecer um rosto humano a partir de uma fotografia em preto e branco. Reduzir a foto a um *array* de números "fino" o bastante para permitir o reconhecimento do rosto exigiria cerca de 1000 por 1000 pontos, ou mais ainda. Isto sugere que o reconhecimento de um rosto, ou outra imagem igualmente complexa, depende de um julgamento feito sobre cerca de um milhão de valores de brilho. Processar este número de valores obviamente levará algum tempo. Será que os humanos realmente reconhecem um rosto, ou outras imagens, na base de exame dos valores de brilho de um vasto número de pontos? Pensando um pouco sobre o assunto, conclui-se facilmente que rostos são reconhecidos na base de suas "características", tais como o tipo de cabelo, a cor dos olhos, a forma da face etc. . . e isto é verdade em todos os tipos de reconhecimento, exceto, possivelmente, os mais simples. Imagens complicadas são reconhecidas em termos da existência ou natureza de um número relativamente pequeno de características.

Esta idéia de que o reconhecimento é baseado em "características" é muito poderosa. Em geral, o problema de reconhecimento pode ser subdividido em dois estágios.

extração das características   classificação

imagem   características   reconhecimento

O primeiro estágio consiste na extração das características, ou seja, verificação da quantidade de cabelo no rosto; em seguida o segundo estágio consiste na classificação da imagem com base nas suas características.

O único problema deste esquema é que, em geral, não se sabe conscientemente quais as características usadas para os humanos reconhecerem objetos, sendo que, na melhor das hipóteses, estas características podem ser estimadas. Como saber se a forma de uma face é considerada quando reconhecemos um rosto? Não será, pelo contrário, que o fato de uma face ser reconhecida que faz com que ela seja vista como uma forma diferente? Apesar da existência deste problema, a idéia de uma característica é uma das mais poderosas em reconhecimento de padrões e será examinada posteriormente.

Ajustando ao Gabarito

Ao invés de continuar considerando idéias abstratas, chegou a hora de pensar em coisas práticas. Considere o problema de reconhecer as letras do alfabeto. Um esquema muito simples consistiria em estabelecer algumas imagens correspondentes a exemplos ideais das letras de A a Z e, quando uma nova imagem for apresentada, pode ser comparada com cada um destes "protótipos" e classificá-la como sendo a letra com a qual "se pareça mais". Os problemas deste tipo de método consistem no estabelecimento dos protótipos e na escolha da maneira de medir a semelhança de duas imagens.

Para as letras do alfabeto, o problema do estabelecimento dos protótipos é relativamente fácil. Por exemplo, a letra A poderia ser definida como

```
00001000
00010100
00100010
00100010
00111110
00100010
00100010
00000000
```


e as outras letras do alfabeto de forma semelhante. A medida da semelhança pode ser feita usando uma idéia da estatística tradicional, o do coeficiente de correlação. Entretanto, pode-se evitar uma matemática mais complicada usando-se um pouco de bom senso. Se uma imagem corresponder exatamente ao seu protótipo, terá uns exatamente nos mesmos lugares, assim, pode-se medir a semelhança contando quantos uns estão nos mesmos lugares em ambas as imagens. Há ainda outra maneira de calcular o número de uns coincidentes que, a princípio, pode parecer algo "confusa", mas que fará muito sentido posteriormente. Multiplicando-se os valores em todos os pixels do protótipo pelos valores dos pixels correspondentes na imagem e somando-se os resultados, obtém-se o número de uns coincidentes. Em BASIC, admitindo que os valores do protótipo estejam armazenados no *array* M e os da imagem no *array* A, isto pode ser calculado por

```
R=0
FOR I=1 TO tamanho da imagem
FOR J=1 TO tamanho da imagem
R=R+M(I,J)*A(I,J)
NEXT I
NEXT J
```

Isto funciona porque M(I,J)*A(I,J) somente dará 1 se *ambos* os pixels forem iguais a 1. Somando os resultados destas multiplicações para todos os pares de pixels dá o total de locais onde ambas as imagens são iguais a 1.

Esta medida pode parecer satisfatória, mas há um senão, pois ela depende do número de uns do protótipo. De fato, o valor máximo que ela pode atingir é exatamente o número de uns do protótipo. Assim, esta medida de semelhança leva em consideração não somente quão semelhantes são as duas imagens, mas também quanto elas têm de preto e branco. A solução para este problema é fácil: obtenha dois novos totais, o número de uns no protótipo e o número de uns na imagem e use-os para remover a dependência dos resultados no brilho geral das imagens. Isto é feito pela fórmula seguinte:

$$\text{medida de semelhança} = \frac{(\text{número de uns coincidentes})^2}{(\text{número de uns no protótipo})*(\text{número de uns na imagem})}$$

Esta fórmula pode parecer um pouco complicada, mas uma rápida consideração mostrará o que está acontecendo nela. O fato mais importante a observar é que, se as duas imagens coincidirem, a medida de semelhança dada por esta fórmula dará 1. Qualquer discordância entre as duas imagens reduzirá este valor (Para os interessados nestas coisas, esta fórmula calcula o quadrado da correlação entre as duas imagens).

Este método de reconhecimento de padrões é usualmente chamado de ajuste ao gabarito, sendo os protótipos os gabaritos que serão usados para decidir a qual classe uma imagem pertence.

Um Programa para Reconhecimento de Letras

O programa seguinte é um exemplo de ajuste ao gabarito aplicado ao reconhecimento de caracteres. Da maneira em que está, pode somente distinguir a letra A de quase todas as outras, mas pode ser facilmente estendido para processar todas as letras do alfabeto.

```
10 DIM M(2,8,8)
20 DATA 00001000
30 DATA 00010100
40 DATA 00100010
50 DATA 00100010
60 DATA 00111110
70 DATA 00100010
80 DATA 00000000
500 DIM A(8,8)

1000 K=1
1010 GOSUB 2000
1020 GOSUB 3000
1030 GOSUB 4000
1040 K=1
1050 GOSUB 5000
1060 GOTO 1030
```



```
2000 FOR I=1 TO 8
2010 READ D
2020 FOR J=1 TO 8
2030 M(K,I,9-J)=D-INT(D/10)*10
2040 D=INT(D/10)
2050 NEXT J
2060 NEXT I
2070 RETURN

3000 PRINT "GABARITO ";K
3010 FOR I=1 TO 8
3020 FOR J=1 TO 8
3030 PRINT M(K,I,J);
3040 NEXT J
3050 PRINT
3060 NEXT I
3070 PRINT
3080 RETURN

4000 FOR I=1 TO 8
4010 PRINT "LINHA ";I;
4020 INPUT LINE
4030 FOR J=1 TO 8
4040 A(I,9-J)=LINE-INT(LINE/10)*10
4050 LINE=INT(LINE/10)
4060 NEXT J
4070 NEXT I
4080 PRINT
4090 FOR I=1 TO 8
4100 FOR J=1 TO 8
4110 PRINT A(I,J);
4120 NEXT J
4130 NEXT I
4140 PRINT
4150 RETURN

5000 R=0
5010 C=0
5020 B=0
5030 FOR I=1 TO 8
5040 FOR J=1 TO 8
5050 R=R+M(K,I,J)*A(I,J)
5060 C=C+M(K,I,J)
5070 B=B+A(I,J)
5080 NEXT J
5090 NEXT I
5100 PRINT (R*R)/(C*B)
5110 RETURN
```

Um *array* tridimensional M é definido na linha 10, e usado para armazenar os gabaritos. Embora neste programa somente um gabarito seja armazenado em M(1,I,I,), outros podem facilmente ser acrescentados. A imagem real que corresponde ao gabarito de A é armazenada nas declarações DATA das linhas 20 — 90. O *array* A é usado para armazenar a imagem de teste. A sub-rotina 2000 lê a definição do gabarito em M, sendo muito fácil adaptar esta para ler mais de um gabarito. A sub-rotina 3000 imprime o padrão correspondente a qualquer gabarito, de acordo com o valor de K, por exemplo, K = 1 imprime o primeiro gabarito. A sub-rotina aceita uma imagem digitada no teclado no mesmo formato 8x8 do gabarito, sendo esta imagem-teste também exibida na tela para que o usuário possa compará-la com o gabarito. Finalmente, a sub-rotina 5000 calcula a medida de semelhança descrita na última seção.

Avaliando o Ajuste ao Gabarito

Se uma letra A perfeita for dada ao programa para reconhecer, a semelhança alcançará seu valor máximo de 1. Entretanto, é importante observar que "perfeito" aqui indica somente que a letra A foi dada exatamente igual ao gabarito. Observando a Figura 6.1, pode-se ver que a semelhança de uma letra B (ao A) dá somente 0,329, o que provavelmente significa que ela não seria classificada como uma letra A! (ou seja, quase certamente, daria um valor maior se comparada com o gabarito de B). Entretanto, observando a Figura 6.2, pode-se ver o resultado para um A distorcido. A semelhança resultante, de 0,62 é ainda maior do que o resultado para a letra B, mas pode ser insuficiente para classificá-la como um A.

Em geral, o ajuste ao gabarito funciona surpreendentemente bem para um enfoque assim simples. Entretanto, a forma de programa usado sofre de um importante defeito. Considere a saída do programa para uma imagem dada que seja um A "perfeito", desenhado ajustado à margem esquerda e na base da grade. Ela teria exatamente o mesmo formato que o gabarito da letra A, mas não se ajustaria bem a ele por estar em uma posição ligeiramente diferente. Para um exemplo mais extremo, considere a letra A desenhada inclinada, ou mesmo


```
00111110          00000000
00100010          00000000
00100010          00001000
00111110          00011000
00100010          00100010
00100010          01000100
00111110          01111110
00000000          00100010

resultado = 0,329   resultado = 0,6213
```

Figura 6.1 Letra B   *Figura 6.2* Letra A distorcida

de cabeça para baixo. A forma continua sendo a mesma, mas o ajuste ao gabarito será considerado muito ruim!

Correlação Cruzada

Há duas soluções possíveis para o problema das posições relativas do gabarito e da imagem. Primeiro, poder-se-ia arranjar que a imagem fosse colocada em uma posição fixa, com uma orientação também fixa. Por exemplo, seria possível providenciar para que todas as letras ficassem em pé e colocadas no meio do *array*. Este método simples não funciona para muitos problemas práticos, pois, para colocar uma letra em pé é necessário saber qual letra é, o que presume a resposta buscada pelo reconhecimento!

A segunda solução, que é melhor, consiste em obter a correlação entre a imagem e um conjunto de gabaritos que representem a letra A colocada em diferentes posições e orientações, tomando finalmente o maior valor de correlação obtido como a medida de semelhança. Isto parece muito razoável do ponto de vista do senso comum, pois, para saber se uma figura é semelhante a outra deve-se movê-la até que se ajuste o melhor possível. A série de valores geradas do coeficiente de correlação é tecnicamente conhecida como a função de "correlação cruzada", mas a origem deste nome não deve constituir preocupação.

Resumindo: o método completo de ajuste ao gabarito consiste do cálculo do coeficiente de correlação entre o gabarito

e a imagem para todas as posições e orientações possíveis, sendo o máximo valor obtido tomado como valor da medida de semelhança entre as duas e, finalmente, a imagem é classificada como correspondente ao gabarito mais semelhante a ela.

Este método funciona bem com problemas tais como reconhecimento de caracteres, mas, em geral, sofre do problema de ser muito lento. Afinal, calcular o coeficiente de correlação entre duas imagens para todas as posições e orientações possíveis leva muito tempo. Para o reconhecimento de caracteres, o cálculo pode ser acelerado um pouco admitindo-se que as imagens estejam em pé, mas, mesmo assim, o método continua lento.

Características e Níveis de Cinza

E onde é que a discussão feita anteriormente sobre características entra neste método dos gabaritos para o reconhecimento de padrões? A resposta é que cada gabarito corresponde a uma característica! Por exemplo, o gabarito da letra A seria usado para medir uma característica que poderia ser chamada de "A-eza". Quanto maior for o valor da medida de semelhança da imagem com este gabarito, mais esta possui de A-eza! Esta descrição pode parecer um pouco estranha, já que o que se procura com este gabarito é reconhecer as letras, mas, imagine que se deseja reconhecer palavras constituídas por estas letras. Cada palavra poderia ser reconhecida por possuir uma lista de características, cada uma delas uma letra em particular. Em geral, qualquer padrão pode ser reconhecido através da definição de um número de gabaritos correspondentes às características que se procura detectar. A classificação final do padrão seria feita com base nas características *ausentes* da imagem.

Embora esta idéia possa parecer limitada pela restrição do ajuste ao gabarito às imagens de dois tons, é fácil estendê-la para incluir imagens com tonalidades de cinza, ou mesmo imagens coloridas. A única diferença nos cálculos é que os números nos *arrays* deverão poder assumir valores diferentes de zero e um. Os coeficientes de correlação resultantes continuarão a medir a semelhança entre as duas imagens e a classificação final poderá ainda ser feita com base em que gabarito melhor se assemelha à imagem dada.

Características de Reconhecimento

A idéia de uma característica é tão fundamental que é difícil imaginar o reconhecimento de padrões sem um primeiro estágio de "detecção de características" ou de "medida de características". O uso de gabaritos para reconhecer imagens é tão simples e direto que parece desnecessário imaginar os resultados do ajuste de um gabarito a uma imagem como uma medida de uma característica da imagem. Este é o caso principalmente no reconhecimento de letras, pois cada gabarito, no caso, representa um dos padrões possíveis, ou seja, as letras de A a Z, na sua totalidade. A razão pela qual o reconhecimento, neste caso, é tão simples, é que ele é baseado na presença ou ausência de uma única característica. Assim, uma letra A é reconhecida por possuir mais A-eza do que B-eza, C-eza etc... (onde A-eza é o grau de semelhança da imagem com o gabarito do A, e da mesma forma para as outras letras). Em um problema mais geral, a situação exigiria a consideração de mais de uma característica, e, até o momento, não foram discutidas maneiras de atacar este problema mais difícil.

Espaço das Características

A chave para a compreensão de como os padrões são classificados quando possuem mais do que uma característica pode ser representada pela expressão "pesar a evidência", o que é exatamente o que os computadores têm que fazer, da mesma forma que os humanos! Tendo sido feitas medidas em, digamos, duas características de um padrão, o problema seguinte consistirá em reduzir os dois números produzidos a um só número, a medida da semelhança entre o padrão dado e o exemplo ideal. Seja o problema que um biólogo tem quando procura identificar a qual mamífero pertence um determinado crânio, que é um exemplo real, pois os biólogos freqüentemente desejam identificar o que algum animal comeu a partir dos restos de sua alimentação. Duas características óbvias no reconhecimento são o comprimento do alto do crânio e sua largura máxima, medidas que, conjuntamente, dão uma idéia grosseira da forma do crânio. Para se ter uma distinção mais refinada seria sempre possível incluir outras características, mas, para maior sim-

plicidade, serão consideradas agora somente estas duas. Colecionando-se medidas em dois tipos de crânio, por exemplo, de lontras e de *minks*, virá o problema de identificar o significado das colunas de números resultantes — o que nunca é uma tarefa fácil. A solução óbvia consiste em desenhar um gráfico das medidas, como mostra a Figura 6.3, onde cada estrela indica um par de medidas de uma lontra e cada círculo um par correspondente de um *mink*. Com um pouco de sorte, cada conjunto de medidas se agrupará em uma região do gráfico, ou seja, haverá uma área no diagrama onde as medidas de crânios de lontras tendem a cair e, da mesma forma, outra área para as medidas dos *minks*. Pode-se ver que este é o caso na Figura 6.3. Se tal agrupamento não ocorrer, não há nada que se possa fazer — isto simplesmente indicará que as características medidas não contêm informação suficiente para distinguir os dois tipos de objetos. Entretanto, se os dois grupos estiverem bem separados, este fato será suficiente para classificar qualquer novo objeto. Executando-se medidas das duas características, um novo objeto poderá ser incluído no diagrama e, se sua identidade não for conhecida, é razoável supor que ele pertença ao grupo do qual mais se aproxime no diagrama. Esta é uma idéia muito boa, e resulta em um grande número de reconhecimentos corretos, mas dificilmente pode ser considerada uma técnica fácil de se aplicar.

Se a sorte for suficiente para encontrar duas características que separem os dois grupos de maneira suficientemente clara para que seja possível traçar uma linha de separação entre eles, pode-se usar um método alternativo de decidir a qual grupo pertence um novo objeto, simplesmente admitindo que pertença ao grupo que estiver do mesmo lado da linha onde suas medidas o colocarem.

Para transformar isto em um método utilizável, tudo o que é necessário é traduzi-lo em termos matemáticos. Denotando as duas medidas de características por X1 e X2, o lado da reta em que o ponto cai é dado pelo sinal de W1*X1+W2*X2. O ponto cairá de um lado da reta se este sinal for positivo, e do outro lado, se for negativo. Mas, afinal de contas, o que serão W1 e W2? W1 e W2 definem a posição e a orientação da reta e, variando-se seus valores, a reta se moverá no diagrama. Obviamente, para usar a equação acima para classificar um novo objeto, é necessário, inicialmente, determinar os valores de W1 e de W2 que melhor correspondam a uma reta que divida os dois grupos de medidas. Isto pode parecer algo complicado, mas, admitindo-se que esta reta corresponde aos valores W1=2 e W2=—2, uma medida que dê X1=3 e X2=2 resultaria em um valor (de 2) positivo para W1*X1+W2*X2, o que significaria que o objeto que gerou estas medidas deveria ser classificado como um exemplo do grupo que ficou no lado positivo da reta.

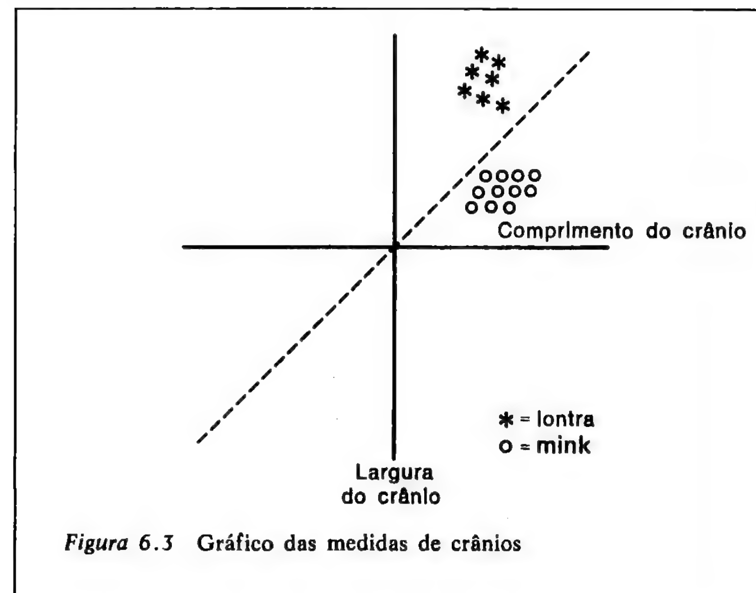


Figura 6.3 Gráfico das medidas de crânios

Recapitulando:

(1) Plotando em um diagrama alguns exemplos de medidas de objetos identificados dos dois grupos, pode ser que estas medidas se agrupem de uma forma tal que os grupos resultantes possam ser separados por uma reta entre eles.

(2) A reta que melhor separe os dois grupos pode ser definida matematicamente de uma maneira que corresponda a valores particulares de duas constantes W1 e W2.

(3) Novos objetos podem ser classificados, ou reconhecidos, através do cálculo de W1*X1+W2*X2, que indica se o ponto cai do lado "positivo" ou do lado "negativo" da reta.

A importante questão que não foi ainda respondida é *como* determinar a reta que melhor separe os dois grupos. Existem métodos que podem determinar (de uma maneira bastante complicada) a reta ótima em um só cálculo, mas tais métodos formam mais parte da estatística teórica do que de AI ou de reconhecimento de padrões. O método que será apresentado leva a um enfoque possivelmente geral ao aprendizado do reconhecimento.

Encontrando uma Reta pelo Aprendizado

Por um momento, admita que já existem medidas de exemplos dos dois grupos. Dando valores arbitrários a W1 e a W2, é possível calcular W1*X1+W2*X2 para cada objeto do conjunto de exemplos. Pode-se, também, admitir arbitrariamente que o grupo 1 está do lado positivo da reta e que o grupo 2 está do lado negativo e, assim, podem-se comparar as previsões da reta arbitrária construída com os grupos dos quais efetivamente vem cada objeto. É pouco provável que os resultados obtidos sejam bons, pois, afinal de contas, atribuir valores arbitrários a W1 e a W2 corresponde a traçar uma reta qualquer no diagrama! Entretanto, usando os resultados obtidos, que mostram se os objetos foram bem ou mal classificados, pode-se tentar ajustar os valores de W1 e de W2 para melhorá-los (veja a Figura 6.4). Isto é que é aprender! Em seguida, esta idéia será discutida com maior profundidade.

Se o cálculo de W1*X1+W2*X2, para um objeto em particular, classificá-lo corretamente, então os valores atuais de W1 e de W2 estarão corretos para este objeto. Entretanto, se ele for classificado no grupo errado, será necessário ajustar os valores de W1 e de W2, o que poderá ser feito de muitas maneiras, todas elas procurando levar a linha correspondente a W1 e W2 a uma posição que chegue o ponto incorretamente classificado mais perto do lado certo. A regra mais simples a usar consiste em simplesmente mover a reta por uma distância fixa, na direção que colocar o ponto incorretamente classificado mais perto de seu lado correto. Matematicamente, esta correção é feita da seguinte maneira:

(1) Se o objeto estiver, na realidade, no grupo 1 (o grupo positivo), então

W1(novo) = W1 + a*X1
W2(novo) = W2 + a*X2

e, se o objeto estiver, na realidade, no grupo 2 (o grupo negativo), então

W1(novo) = W1 — a*X1
W2(novo) = W2 — a*X2

onde "a" é uma constante que controla a velocidade na qual os objetos são "aprendidos".

(2) Se este ajuste for executado para cada um dos objetos disponíveis, espera-se que, a cada repetição do processo, o comportamento da classificação melhore. Repe-

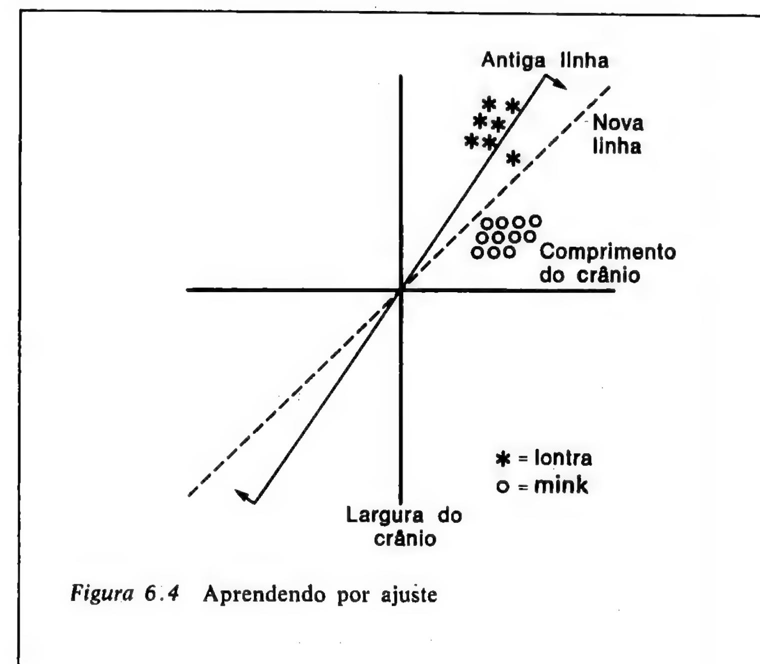


Figura 6.4 Aprendendo por ajuste

tindo-se continuamente o processo, haverá um momento em que não será mais possível nenhum melhoramento: o aprendizado do reconhecimento do conjunto de objetos terá sido conseguido. Os valores finais de W1 e de W2 obtidos poderão, então, ser usados para tentar a classificação de qualquer objeto novo que apareça.

O processo descrito acima, embora simples, tem todos os ingredientes do aprendizado. O conjunto inicial de objetos, cuja identidade é conhecida, é usado como um "conjunto de treinamento". O classificador, ou reconhecedor, dado por W1*X1 + + W2*X2 é "ensinado" a reconhecer cada um dos objetos conhecidos, via a tentativa de classificação e a correção, se estiver incorreta. Logo que tiver dominado o conjunto de treinamento, o reconhecedor poderá ser usado para classificar objetos desconhecidos. Este tipo de aprendizado é freqüentemente chamado de "aprendizado supervisionado", devido à necessidade de um "professor" para ajustar os valores de W1 e de W2 durante o treinamento. É possível, também, realizar o treinamento sem supervisão, o que será descrito posteriormente.

Todas as idéias descritas até agora podem ser estendidas para o caso em que mais de duas características são medidas. Por exemplo, no caso de três características, a classificação será dada por

$$W1*X1 + W2*X2 + W3*X3$$

Em geral, N características precisarão de N valores de W. Outra maneira de imaginar os valores W é considerá-los como fatores de ponderação nas fórmulas. Assim, o tamanho e o sinal de cada W mostrará quanta importância é dada a cada característica, no reconhecimento do objeto.

Para os que estiverem pensando em como proceder quando houver mais de dois grupos de objetos a reconhecer, a resposta é que eles deverão ser tratados dois a dois!

O Perceptron

A descrição do aprendizado feita anteriormente foi baseada na idéia da medida de características e no treinamento posterior de um reconhecedor. Assim, estando as características devidamente

escolhidas, tudo funciona como foi descrito, mas, como escolher quais características usar? O que acontecerá se a escolha não recair em boas características? Tais questões são de difícil resposta geral e, sem dúvida, o uso de um humano para escolher as características que o reconhecedor usará parece uma fuga do problema principal! Existe uma maneira de deixar que o programa ou a máquina escolha as características, mas, para examiná-la, será necessário antes estender a definição de "característica". Até agora, "característica" significou uma medida que "faz sentido" para um humano, entretanto, por que deverão as características usadas para reconhecer padrões fazerem sentido para os humanos? Talvez todas as características, cujo uso os humanos defendem, sejam simplesmente racionalizações escondendo aquelas que são realmente usadas! Certamente, nas profundidades do cérebro humano haverá características mais primitivas que são usadas para reconhecer coisas que não precisam corresponder a algo que tenha um nome. Esta idéia mais geral de uma característica pode ser mais facilmente compreendida após a descrição de uma máquina (ou um programa) que constrói suas próprias características, o Perceptron.

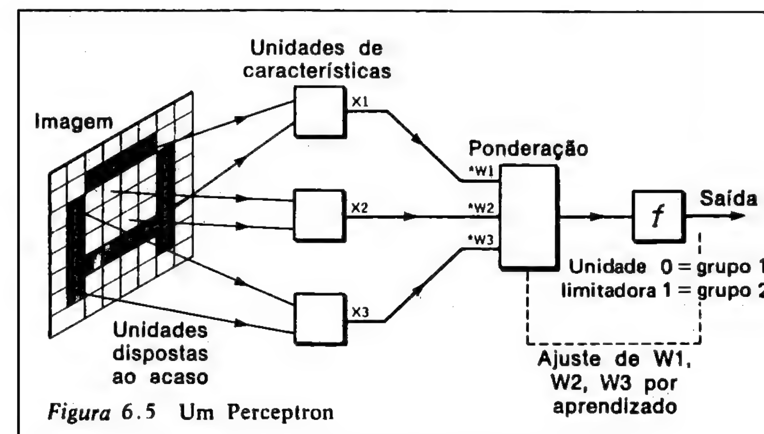


Figura 6.5 Um Perceptron

O Perceptron é um reconhecedor que aprende e que funciona de uma maneira muito semelhante ao programa de aprendizado supervisionado descrito anteriormente. A saída de vários aparelhos de medida de características é ligada a uma unidade

que calcula o equivalente a W1*X1 + W2*X2 + etc... A saída desta unidade é, então, ligada a uma unidade limitadora que, de fato, escolherá o grupo ao qual o padrão pertence. Tudo isto pode ser visto na Figura 6.5. O que é diferente no Perceptron é a maneira de selecionar e medir as características. A Figura 6.5 mostra que cada um dos dispositivos de medida de características tem a forma de várias ligações que medem o brilho da imagem em alguns pontos. Estas medidas são processadas, resultando num 0 ou um 1, após somadas e passadas por uma unidade limitadora, cuja saída será 1 se um padrão determinado estiver presente em cada um dos pontos ligados e 0 em caso contrário. As posições das ligações são arbitrárias, sendo, de fato, vantajoso que sejam escolhidas aleatoriamente ao longo de todo o padrão. Um Perceptron deve ser treinado usando um conjunto conhecido, antes que possa atuar e, à medida que seu treinamento progredir, algumas das saídas das unidades de medida de características passarão a ser ignoradas, sempre que os valores correspondentes de W caírem abaixo de um valor muito próximo de zero. As características úteis escolhidas serão aquelas cujos valores de W, ao final do treinamento, forem diferentes de zero, e, como poderia ser esperado, a *maioria* das características será inútil, devido à sua escolha ter sido aleatória. Para superar este problema, um Perceptron real deve ter um número muito grande (maior que 200) de unidades de medida de características, para ter uma boa chance de encontrar características boas! Isto tudo pode parecer muito grosseiro, mas provavelmente está muito perto da maneira de o cérebro humano aprender, pois, afinal de contas, há neurônios suficientes para que alguns sejam desprezados!

O Perceptron é um exemplo útil de uma máquina que aprende. Recebeu muita crítica nos meios acadêmicos, mas é ainda uma boa maneira de executar reconhecimentos práticos. Programas tipo Perceptron e mesmo equipamentos desta natureza são efetivamente usados em algumas máquinas comerciais de reconhecimento de padrões, constituindo o componente principal em reconhecedores da palavra falada, cujos padrões de entrada consistem na quantidade de potência recebida em algumas faixas de freqüência. Isto é que explica o fato de os acessórios de reconhecimento da fala terem que ser ensinados a reconhecer as palavras com as quais irão trabalhar.

Há ainda um ponto a discutir. Os métodos de aprendizado considerados são todos métodos de ensino supervisionado, exigindo um professor que saiba as respostas corretas na fase do aprendizado, o que corresponde a muitas situações de aprendizado humano, mas não a todas. De alguma maneira, às vezes os humanos conseguem aprender sem professor, o que, com algumas rápidas considerações, pode ser facilmente entendido, e não tem mágica nenhuma! Um humano difere das máquinas simples que foram discutidas no fato de que *agem* conforme os resultados dos reconhecimentos feitos, o que permite que, observando os resultados posteriores dos atos, seja possível concluir sobre o acerto ou não do reconhecimento feito e, assim, ajustar devidamente o método de reconhecimento. Até que as máquinas de reconhecimento se tornem um pouco mais ousadas, precisarão de um professor para avaliar suas decisões.

Aqueles que acharem que o Perceptron é uma forma muito limitada de reconhecedor, provavelmente terão razão, pois é difícil imaginar um único Perceptron tendo capacidade para reconhecer uma face humana determinada. Entretanto, por que limitar-se ao uso de um único Perceptron? Uma máquina sofisticada de reconhecimento usaria muitos "bancos" de Perceptron, cada um entregando suas saídas aos outros na forma de características detectadas. À medida que este processamento progride, as características entregues aos bancos posteriores tornar-se-ão cada vez mais complexas e, assim, mais do tipo de características às quais humanos dariam nomes. Por exemplo, o primeiro banco pode reconhecer linhas em uma imagem, o segundo ângulos e linhas interligadas, um banco muito adiante detectaria quadrados e outras formas, e assim por diante, tudo isto, entretanto, longe da capacidade de um microcomputador!

Usos do Reconhecimento de Padrões

Em certos casos, o reconhecimento de padrões já é um fim em si mesmo, como, por exemplo, no caso do reconhecimento de letras, que é um problema importante e útil já que, se for possível reconhecer letras, será também possível eliminar todo o trabalho atualmente executado para transferir textos impressos

para computadores. Assim também, o reconhecimento da fala é uma finalidade importante em si mesma, mesmo quando não utilizada em conjunto com outros *softwares* inteligentes. Estas e outras aplicações importantes resultaram em uma tendência a enfatizar o reconhecimento de padrões como um assunto separado, sem muitas ligações com o restante da AI. Entretanto, parece razoável supor que esta situação mudará quando forem encontradas soluções para os problemas mais simples de reconhecimento de padrões. As áreas de visão e audição artificiais são claramente importantes para uma solução completa e satisfatória do problema mais geral da AI e, no futuro, a ênfase deverá se deslocar para a interação entre AI e reconhecimento de padrões. Há outra parte do reconhecimento de padrões que é também muito importante para AI, se bem que de uma maneira menos óbvia. Por exemplo, um dos principais problemas da implantação dos sistemas de conhecimento baseados em regras consiste do reconhecimento da "condição" que forma parte da regra IF. . .THEN. Esta, que nos exemplos dados no Capítulo 4 era fácil de ser detectada, passará cada vez mais a ser uma parte importante do processo à medida que os programas que raciocinam atacarem áreas mais complicadas. Aí, os métodos de reconhecimento de padrões serão indispensáveis. Os programas de AI do futuro deverão incluir um estágio inicial de reconhecimento de padrões para fornecer os dados com os quais trabalharão.

7 Linguagem

Após breve consideração, é fácil concluir que os computadores passam muito mais tempo processando textos do que "triturando" números, e o mais interessante é que todo este processamento de texto consiste basicamente em transferir caracteres de um lugar para outro, ou seja, quase todos estes "encontros imediatos" entre computadores e linguagens são pouco mais do que armazenamento e recuperação de caracteres. A meta última é, obviamente, melhorar a situação atual até o ponto em que os computadores mostrem algum grau de compreensão para a linguagem humana cotidiana. Os primeiros computadores que o conseguirem serão necessariamente grandes e serão encontrados principalmente em laboratórios de pesquisa. Entretanto, quando o equivalente futuro dos micros atuais conseguir possuir esta habilidade, o mundo da computação ficará *muito* diferente.

Considerar o futuro é divertido, e pode mesmo ser útil na preparação do caminho para os novos usos dos computadores. Entretanto, muito mais do que é feito hoje em dia já pode ser realizado tratando textos como linguagens, mesmo usando métodos bastante simples, disponíveis nos menores micros. Até um certo ponto, o que é feito com os computadores é o que é necessário. Assim, processadores de textos constituem um uso muito óbvio e lucrativo dos computadores, mas muito tempo passou antes que alguém pensasse em escrever programas que executassem correções de ortografia ou de gramática. Parece haver uma barreira para os programadores, sempre que pen-

sam em termos do processamento da linguagem, cujo tratamento é realmente muito difícil se o objetivo for a completa compreensão, mas há muitos objetivos mais fáceis neste caminho. Para aqueles de espírito muito técnico, linguagem pode lembrar a antiga matéria escolar da gramática, assustando-os muito, na verdade sem razão, pois, fora algum jargão novo, não há do que ter medo!

Sintaxe e semântica

Há sempre dois componentes de uma linguagem qualquer: a maneira na qual é escrita — sua sintaxe — e o significado que contém — sua semântica; destes dois, a sintaxe é simultaneamente o mais fácil e o menos interessante. Entretanto, é necessário entender a sintaxe antes de ser possível compreender o significado da linguagem, e também, para muitas aplicações práticas, a compreensão da sintaxe é suficiente para extrair o significado aproximado.

Sintaxe não é nada novo, tendo sido estudada, já, há muito tempo, sob o nome de gramática em todos os níveis, desde a escola primária até a pesquisa pura. O primeiro mito a desfazer é o de que existe um conjunto rigoroso de regras que formam a gramática de qualquer linguagem, o que não é verdade, pois uma linguagem, tal como o inglês ou qualquer outra, é suficientemente sutil para fazer com que nenhum conjunto simples de regras seja suficiente para descrever todas as maneiras em que se pode dizer alguma coisa. Outra idéia importante é que nem tudo o que obedece mesmo às mais simples regras da gramática faz sentido. Por exemplo,

o sonho dorme furiosamente

é uma sentença corretamente formada em português, tendo um substantivo, um verbo e até um advérbio, mas não tem nenhum sentido. "Sonho" é um substantivo, mas não é algo que possa "dormir", que, por sua vez, é um verbo ao qual não é razoável aplicar o advérbio "furiosamente". Em resumo, uma sentença correta mas sem sentido! Isto deve ilustrar o fato de que a sintaxe e a semântica são componentes distintos da linguagem.

Descrevendo a Sintaxe

Antes que seja possível analisar uma sentença em particular, é necessário encontrar uma forma de descrever as maneiras de juntar as palavras. Por exemplo,

mat the on sat cat* (*esteira o na sentou gato*)

é uma versão desordenada de uma sentença em inglês, mas, obviamente, este conjunto de palavras *não é* inglês. O que elimina esta sentença das possíveis em inglês é mais do que o fato de ela não fazer nenhum sentido. É o fato de que ela fere algumas simples regras sobre a ordem em que as palavras devem vir na sentença. A observação importante aqui é que, embora possa haver sintaxe sem semântica, esta última não existe sem a primeira.

Embora seja difícil encontrar um conjunto completo de regras para descrever a língua inglesa, é possível fazer alguma coisa neste sentido. Por exemplo, uma sentença típica tem a forma

sujeito predicado

Em outras palavras, a maioria das sentenças em inglês são sobre alguma coisa — o sujeito — fazendo alguma outra coisa — a ação representada pelo predicado. Na sentença

He jumped (*ele saltou*)

"He" é o sujeito e "jumped" o predicado. As coisas podem, entretanto, ser mais complicadas, pois tanto o sujeito como o predicado podem consistir de mais do que uma palavra. O que é necessário é produzir algumas regras que governem a maneira

* N. T.: Tendo em vista o fato de que toda a descrição de linguagem, feita neste capítulo, se refere à língua inglesa e, ainda mais, de que a organização da maioria das linguagens computacionais segue de perto a organização desta língua, os exemplos serão deixados em inglês, assim como o programa que será apresentado. Logo em seguida a cada trecho em inglês, entretanto, será incluída, entre parênteses e em cursivo, a tradução correspondente, de maneira a permitir ao leitor acompanhar perfeitamente a descrição.

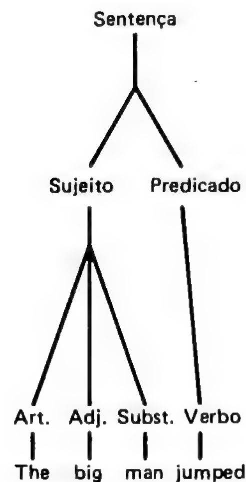
de construir tais frases, de forma a garantir que, se uma sentença for formada reunindo seus elementos na ordem correta, estará também correta. Por exemplo, um sujeito pode ser formado da seguinte maneira:

artigo   adjetivo   substantivo

onde artigo é uma dentre uma lista muito pequena de palavras, tais como "a" (*um, uma*), "the" (*o, a*), ou "some" (*uns, umas, algum, alguma*), e um adjetivo descreve o substantivo ao qual estiver associado. Com isto, a sentença anterior pode ser estendida para

The big man jumped   (*o homem grande saltou*)

Uma maneira de "ver" a estrutura da sentença e as regras que ela segue consiste novamente de usar o diagrama em árvore



Usando este diagrama é possível ver como é construída esta simples sentença. Note que as palavras somente aparecem na base da árvore, enquanto que, nos outros níveis, aparecem palavras que descrevem partes da sentença.

Análise gramatical*

Pode-se imaginar a compreensão da sintaxe de uma sentença como a busca da árvore de sintaxe que a descreva, o que é conhecido como fazer a "análise gramatical" da sentença, que constitui um pré-requisito para poder compreendê-la. Em geral, executar a análise gramatical de uma sentença em inglês é muito difícil e somente pode ser conseguido por um processo seqüêncial de teste de possibilidades, o que seria uma tarefa muito extensa para um micro programado em BASIC. Uma maneira de fazer a análise gramatical consiste em começar no alto da árvore de sintaxe e tentar ajustar as diferentes partes constituintes da sentença, o que é conhecido como analisar de "cima-para-baixo". Como é óbvio, o processo contrário, começando da base da árvore, é conhecido como análise de "baixo-para-cima". A dificuldade da análise gramatical é que, simplesmente, há possibilidades demais para testar em uma sentença típica e o simples ajuste da base da árvore às palavras da sentença já é uma tarefa formidável, que exigiria uma lista de todos os substantivos e verbos que o programa poderia encontrar, além, obviamente, dos adjetivos e advérbios.

Para animar um pouco, vale a pena mencionar o fato de que as linguagens computacionais são muito mais simples e que muitos compiladores começam o processo de tradução da linguagem de alto nível em liguagem de máquina exatamente executando a análise gramatical de cada declaração do programa.

A Geração de Linguagem

A importância das árvores de sintaxe, no caso dos pequenos computadores, está mais relacionada com a geração de sentenças corretas, pois são ideais do ponto de vista de garantir que a sintaxe de uma sentença seja clara para um humano, mas os computadores não trabalham facilmente com diagramas! Para superar esta dificuldade é necessário encontrar uma maneira

* N. T.: Análise gramatical em inglês é "parsing", palavra que muitos dos que trabalham na área de informática preferem não traduzir. Como muitos outros traduzem, o tradutor preferiu seguir estes últimos.

não-gráfica de representar as regras da sintaxe, sendo a solução mais comum o uso de "regras de produção", que, para uma sentença simples, podem ser:

S→frase nominal   frase verbal
onde:
frase nominal→artigo   adjetivo   substantivo
frase verbal→verbo

onde o sinal "→" pode ser lido como "é um(a)". Estas regras são chamadas de regras de produção, porque mostram como um elemento de sintaxe pode ser "produzido" a partir de outro.

Há vários problemas com esta notação. O primeiro é que é difícil distinguir entre coisas que serão mais detalhadas posteriormente e coisas que irão ser diretamente substituídas por palavras. Por exemplo, o termo "frase nominal" deveria ser mais detalhado, mas "substantivo" e "verbo" poderão ser diretamente substituídos por palavras de sua classe. Para mostrar esta diferença, serão usados sinais de "menor que" e "maior que" <> para indicar os itens que terão detalhamento posterior. Um outro problema é que uma parte da sentença, tal como a frase nominal, pode ter um grande número de diferentes definições. Este problema é resolvido usando a barra vertical "|" para indicar "escolha um de". Por exemplo,

<frase nominal>→pronome|artigo   adjetivo   substantivo

significa que a <frase nominal> pode ser tanto um pronome tal como "he" (*ele*) ou "she" (*ela*), etc... quanto uma estrutura como a descrita anteriormente. Uma ilustração melhor pode ser dada por um detalhamento da frase verbal

<frase verbal>→ verbo   advérbio nada (<CV>

o que significa que a <frase verbal> é um verbo tal como "jump" (*saltar*), seguido de um advérbio ou de "nada". O uso de "nada" é simplesmente uma maneira de dizer que um advérbio pode seguir a um verbo ou não! O interessante elemento extra é <CV>, que significa "complemento verbal". Sua definição é

<CV> preposição <frase nominal> <CV>|nada

A primeira parte desta definição é simples, já que uma preposição é algo tal como "on" (*sobre*), "under" (*sob*) etc..., mas o restante da definição é interessante. A ocorrência da <frase nominal> na definição é bastante surpreendente, mas <CV> ocorrer em sua própria definição...! O segredo da compreensão disto está no "|nada", que diz que pode haver vários <CV> em sua sentença, sendo possível acompanhar uma "preposição <frase nominal>" por outra, ou também escolher o "nada" e parar por aí. Como um exemplo deste tipo mais elaborado de sentença, seja a versão correta da frase desordenada que foi citda anteriormente

The cat sat on the mat (*o gato sentou sobre a esteira*)

Neste caso a <frase verbal> é "sat on the mat", onde o verbo é "sat" e o <CV> é "on the mat", que pode ser ainda subdividido na preposição "on" e na <frase nominal> "the mat". A sintaxe resultante pode ser vista na Figura 7.1.

Usando este conjunto de definições é possível escrever um programa que pode gerar textos em inglês de forma bastante convincente. O algoritmo do programa é intimamente relacionado às definições de sintaxe. De fato, é tão intimamente relacionado que seria bem possível escrever um programa que lesse uma lista de definições de sintaxe e gerasse sentenças com sentido, e corretas de acordo com as regras lidas. Entretanto, esta seria uma tarefa mais ambiciosa do que a do programa considerado a seguir.

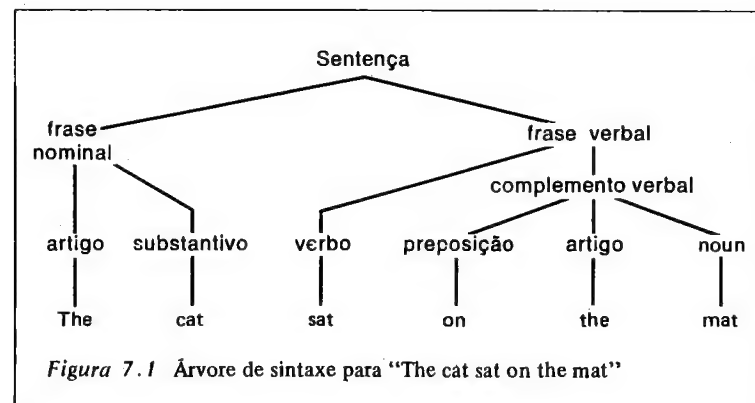


Figura 7.1 Árvore de sintaxe para "The cat sat on the mat"

O princípio básico do programa gerador de sentença é que estas são construídas da esquerda para a direita e há uma sub-rotina correspondente a cada item delimitado por sinais de menor e maior. Cada sub-rotina pode chamar outra sub-rotina, fornecer uma palavra do tipo correto ou não fazer nada, o que é o equivalente, no programa, do "nada".

O Programa da Conversa do Computador

Um pequeno programa que usa as regras de sintaxe dadas anteriormente e agora apresentado, sendo as próprias regras colecionadas na Tabela 7.1.

TABELA 7.1

<SENTENÇA> → <frase nominal> <frase verbal>

<frase nominal> → ARTIGO ADJETIVO SUBSTANTIVO

<frase verbal> → VERBO ADVÉRBIO <CV> NADA

<CV> → PREPOSIÇÃO <frase nominal>

Pode parecer surpreendente o fato de o programa ser tão curto, mas a maior razão para isto é o seu pequeno vocabulário, de somente 19 palavras! Mesmo assim, é divertido ver o que ele tem a dizer. Dois exemplos de saídas suas são

> The tiny bugs runs to the big computer (*o besourinho corre para o computador grande*)

> A tiny computer prints quickly under a big computer (*um pequeno computador imprime rapidamente sob um grande computador*)

nenhuma das quais é muito profunda, apesar de ambas terem charme ingênuo.


```
  10 GOSUB 1000
  20 GOSUB 2000
  30 GOSUB 3000
  40 PRINT S$
  50 STOP

1000 ART=2
1010 DIM A$(ART)
1020 A$(1)="A"
1030 A$(2)="THE"

1100 NOUN=4
1110 DIM N$(NOUN)
1120 N$(1)="COMPUTER"
1130 N$(2)="PRINTER"
1140 N$(3)="PROGRAM"
1150 N$(4)="BUG"

1200 ADJ=3
1210 DIM D$(ADJ)
1220 D$(1)="BIG"
1230 D$(2)="POOR"
1240 D$(3)="TINY"

1300 VERB=4
1310 DIM V$(VERB)
1320 V$(1)="WRITES"
1330 V$(2)="RUNS"
1340 V$(3)="DEBUGS"
1350 V$(4)="PRINTS"

1400 ADVERB=3
1410 DIM B$(ADVERB)
1420 B$(1)="SLOWLY"
1430 B$(2)="FAST"
1440 B$(3)="QUICKLY"

1500 PREP=3
1510 DIM P$(PREP)
1520 P$(1)="ON"
1530 P$(2)="TO"
1540 P$(3)="UNDER"

1600 S$=""
1610 RETURN

2000 REM NOUN PHRASE
2010 REM ARTICLE
2020 S$=S$+" "+A$(INT(RND(0)*ART)+1)
2030 REM ADJECTIVE/NULL
```

```
2040 IF RND(0)>.5 THEN S$=S$+" "+
     D$(INT(RND(0)*ADJ)+1)
2050 REM NOUN
2060 S$=S$+" "+N$(INT(RND(0)*NOUN)+1)
2070 RETURN

3000 REM VERB PHRASE
3010 REM VERB
3020 S$=S$+" "+V$(INT(RND(0)*VERB)+1)
3030 REM ADVERB/NULL
3040 IF RND(0)>.5 THEN S$=S$+" "+
     B$(INT(RND(0)*ADVERB)+1)
3050 GOSUB 4000
3060 RETURN

4000 REM PREPOSITIONAL PHRASE
4010 REM OR NULL
4020 IF RND(0)>.5 THEN RETURN
4030 REM PREPOSITION
4040 S$=S$+" "+P$(INT(RND(0)*PREP)+1)
4050 REM NOUN PHRASE
4060 GOSUB 2000
4070 RETURN
```

A sub-rotina 1000 organiza as listas de palavras. Há um *array* para cada tipo de palavra usada em uma sentença, e é fácil incluir mais palavras no vocabulário. Cuidado, porém, para pôr as palavras nas listas corretas! A sub-rotina 2000 forma uma frase nominal extraindo uma palavra ao acaso da lista de artigos, selecionando opcionalmente da lista de adjetivos e então selecionando uma palavra da lista de substantivos. A sub-rotina 3000 trabalha aproximadamente da mesma maneira, exceto por chamar a sub-rotina 4000 para fornecer um complemento verbal e, quando isto acontece, a sub-rotina 2000 é novamente chamada para fornecer outra frase nominal. Esta nova utilização da sub-rotina 2000 mostra o poder do método geral de escrita de sub-rotinas para gerar cada parte da sentença, pois ela gera uma <frase nominal> seja para uso direto no início da sentença, seja como parte de um complemento verbal, gerada por outra sub-rotina.

Da mesma forma que todos os outros exemplos neste livro, este também foi escrito no Microsoft BASIC, o que deve tornar fácil sua conversão para outras versões de BASIC. Embora seja um programa muito divertido, está longe de ser uma versão definitiva, sendo, na realidade, somente o ponto de partida para suas experiências com geração automática de textos legíveis. Experimente estender o vocabulário e as regras de sintaxe para que as sentenças façam maior sentido. Um bom exercício seria melhorar o programa ao ponto em que ele gerasse textos que pudesesm enganar alguém no sentido de achar que tivessem sido escritos por humanos! Uma idéia ainda mais útil seria tentar produzir um programa que gerasse pequenos poemas ou algo semelhante.

Sintaxe e Significado

A sintaxe trata da maneira de reunir as palavras para formar frases corretas em uma determinada linguagem. A semântica trata do significado contido nestas frases sintaticamente corretas. Embora seja possível gerar sentenças em uma linguagem usando as regras de produção, não é garantido que resulte algum significado nestas sentenças. Quando o programa da "conversa de computador" imprime uma sentença, é óbvio que não vai aí nenhuma tentativa de comunicação do computador. Por exemplo, ele pode produzir facilmente uma sentença do tipo

The program runs slowly (*o programa executa lentamente*)

Quando isto aparecer na tela ou na impressora, não significa que o computador esteja criticando o programa, ou será que significa?

Significado Inocente

Se alguém decidir gastar o tempo e o trabalho necessários para aumentar o vocabulário e as regras de sintaxe do programa da conversa de computador, poderá fazê-lo produzir uma ampla gama de sentenças com aspecto natural. Se, em seguida, alguém que não sabe de nada, e com o espírito desprevenido, sentar-se em frente a um terminal ou um computador que esteja executando tal programa, sua reação dependerá do que ouviu falar dos computadores no passado. Entretanto, muitas pessoas irão ler a saída do programa como se esta tivesse algum significado e achar que o computador assumiu a personalidade de alguém confuso e algo louco preso dentro de uma caixa cheia de eletrônica!

O ponto é que, para alguém que saiba como o programa da conversa de computador produz textos, as sentenças obviamente são destituídas de sentido, mas alguém desprevenido poderá fazer uso da grande capacidade que têm os humanos de encontrar sentido em algo que absolutamente não o tem. O resultado desta tendência humana de encontrar sentido em tudo, é que torna muito fácil convencer um usuário inocente de que os computadores têm muito mais capacidade do que efetivamente possuem. É sorte que os especialistas em AI são uma turma honesta, pois, caso contrário seria muito fácil para eles enganarem outras pessoas (inclusive muitos outros especialistas em AI!), fazendo-as acreditarem que conseguiram muito mais do que na realidade alcançaram.

É claro que será necessário considerar um pouco mais cuidadosamente o que torna a linguagem cheia de sentido, mas, antes disto, vale a pena dar uma outra olhada nos métodos para gerar linguagem correta.

Aproximações para Linguagens

Admitindo, inicialmente, desconhecer tudo o que já foi dito sobre a sintaxe, serão examinadas agora outras maneiras de gerar linguagem. Poder-se-ia começar escrevendo um programa BASIC que reunisse as letras do alfabeto de uma maneira completamente ao acaso. A saída de tal programa poderia, vez por outra, produzir alguma palavra ocasional, mas, na maior parte do tempo, somente produziria disparates. Para melhorar a probabilidade de conseguir algo com sentido, a providência mais óbvia consistiria em alterar a freqüência com que cada letra fosse usada. Por exemplo, não é razoável gerar tantos "Z"s quantos "A"s, pois a letra Z aparece muito menos freqüentemente que a A, em inglês. O melhor a fazer seria ajustar a freqüência em que cada letra ocorresse para aquela em que ela ocorre no inglês normal. Para melhorar ainda mais o desempenho do programa, seria necessário, também, assegurar que os pares possíveis de letras também ocorressem com a mesma freqüência com que o fazem no inglês normal, após o que poder-se-ia considerar o conjunto de três, quatro letras etc. . . A medida que conjuntos maiores de letras fossem considerados no programa, a linguagem gerada tornar-se-ia cada vez mais parecida com o inglês normal.

A importância de tudo o que acaba de ser dito é que é possível produzir aproximações cada vez melhores a uma linguagem sem saber nada sobre sua sintaxe ou sua semântica. Tudo o que é necessário fazer é colecionar um conjunto de exemplos e contar as freqüências de ocorrência dos grupos de letras — um trabalho facilmente feito por um computador! Qual será, então, o sentido de estudar a sintaxe, se um simples esquema aleatório também pode produzir linguagem correta?

Significado a Partir da Sintaxe

O valor do conhecimento da sintaxe não vem da sua possibilidade de produzir linguagem correta, mas da forma que fornece um "guia" para o sentido de uma sentença. Dada uma sentença, definida sua sintaxe, resulta uma espécie de "mapa" que pode ser usado para investigar seu significado. Se a sentença for da forma "SUJEITO" "PREDICADO" sabe-se imediatamente que ela diz que o objeto constante do "SUJEITO" está executando a ação definida no "PREDICADO". Ainda mais, conhecendo o tipo de sentença com que se está trabalhando, será possível selecionar a "próxima ação" apropriada. Entretanto, conforme já foi explicado, a tarefa de determinar a sintaxe, ou executar a análise gramatical, de uma sentença, é muito violenta para o BASIC.

Este esquema proposto tem ainda outro defeito, que é mais grave. Fora uma parte muito reduzida do inglês, não existe uma gramática ou uma sintaxe exata, do tipo que vem sendo discutido. Há muitas extensões às árvores de sintaxe simples que têm sido discutidas, mas, até o momento, não foi possível descrever a linguagem humana em toda sua complexidade.

Compreensão Prática

Se o último parágrafo deixou-o desanimado, anime-se, novamente, porque, com um pouco de coragem e muito esforço é possível escrever programas que tenham uma parcela limitada de compreensão e que também respondam apropriadamente. O importante é não querer demasiado, e lembrar sempre que,

em alguma parte do sistema, deve haver um humano desejando ajudar um pobre e confuso computador.

Este seria um projeto muito ambicioso para usar como exemplo e, conforme já foi dito várias vezes, BASIC não é exatamente a linguagem conveniente para tal tarefa. Entretanto, com o que já foi discutido, deve ser possível entender como tal programa limitado poderia ser escrito. Por exemplo, suponha que se deseja um programa que forneça o horário e certas informações sobre as rotas de algumas linhas de ônibus. Para a parte do programa que deve fazer a "interface humana" pode ser adotada a solução tradicional de fornecer um "menu" de opções e apresentar perguntas, em seguida, do tipo "qual é o número do ônibus?" e "qual é o código do destino?" Esta é uma solução fácil, pois evita trabalhar com palavras nas respostas que devem ser dadas pelo usuário, mas, simultaneamente, irá fortalecer a crença do "homem comum" de que computadores somente podem tratar com números! Uma solução algo mais ambiciosa consistiria em permitir que o usuário "conversasse" com o computador, usando um tipo de linguagem limitada, mas natural. Isto poderia ser feito imaginando-se, inicialmente, a gama de sentença que um usuário poderia digitar no terminal e construir um pequeno grupo de regras de sintaxe que pudesse gerá-las. Uma pergunta comum poderia ser

Quando sai o próximo ônibus?

que poderia ser descrita pela sintaxe

<Pergunta> é a <descrição do ônibus>

O item <Pergunta> poderia ser expandido para incluir coisas tais como "quando" ou "onde". A <descrição do ônibus> poderia ser mais detalhada, incluindo diferentes maneiras de especificar qual o ônibus referido. Comparando o que o usuário digitar com uma lista de diagramas possíveis de sintaxe, o programa poderia determinar qual a resposta que deveria dar, ou seja, se uma sentença digitada pelo usuário coincidir com a sintaxe dada anteriormente, o programa deveria dar o horário do ônibus especificado. Este não é um esquema difícil de programar, mas necessita, efetivamente, de um computador de porte médio, com uma boa quantidade de memória, para guardar todo o vocabulário. (Esta solução para a compreensão é conhecida por "sintaxe pragmática").

Boa Vontade e compreensão: Eliza

Para encerrar esta introdução a computadores e linguagem, vale a pena considerar um programa que alcançou alguma notoriedade, o Eliza, escrito há não muito tempo, durante as primeiras experiências com linguagem e AI e que podia manter uma boa conversação com uma pessoa humana sobre uma certa variedade de tópicos. Ele recebeu este nome devido à famosa personagem da peça de Shaw, "Pigmalião". Sua maneira de trabalhar era bastante simples, pois detectava a presença de algumas "palavras-chave" e, alterando os tempos verbais e extraindo cláusulas das mensagens dos usuários, podia "dar a volta" nas sentenaçs e "dispará-las" de volta ao usuário. A versão de maior sucesso deste programa podia conversar sobre quaisquer problemas que o usuário tivesse (ou imaginasse ter) e foi chamada de "Eliza Doctor". Embora este programa fosse muito simples, tinha um grande vocabulário, que, combinado com a maneira que os humanos "lêem significado" na conversa, o tornava muito convincente. Chegou a ser tão convincente que pacientes e douotres "reais" começaram a usá-lo. Na verdade, chegou a ser tão bom que muitos pacientes o preferiam a alguns médicos humanos!! O criador do programa e muitos outros especialistas em AI ficaram muito alarmados com a facilidade com que foi possível fazer um bom número de pessoas comuns e bem-educadas manterem longas e íntimas conversas com um programa completamente sem inteligência!

Os princípios usados pelo Eliza são muito fáceis de descrever. Inicialmente, varre a entrada em busca de algumas "palavras-chave", que, quando detectadas, disparam sempre a mesma ação, que consiste em ou devolver uma mensagem padrão, ou usar parte da entrada para construir uma mensagem. Por exemplo, se o usuário digitasse

Odeio sorvete

o programa detectaria a palavra-chave "odeio" e responderia

Não é bom odiar.

Observe que esta resposta era dada, independentemente do restante da frase. Se isto fosse tudo o que o Eliza fizesse, ficaria muito fácil descobrir que se tratava de um programa, pela quan-

tidade limitada de respostas que daria. Entretanto, para introduzir alguma variação em suas respostas, o programa fazia uso da sentença de entrada. Qualquer sentença que o usuário digitasse seria varrida em busca de certas palavras ou frases tais como "meu" ou "você é" e, caso alguma delas fosse encontrada, seria transformada no oposto apropriado, ou seja, "meu" é transformado em "seu" e "você é" em "eu sou". A finalidade destas transformações simples é devolver ao usuário as sentenças que tenha introduzido, como se estas fossem geradas pelo programa. Por exemplo, recebendo a sentença

Você é um idiota

o computador devolveria

sou um idiota

possivelmente acompanhada de alguns sinais de exclamação ou de interrogação.

Estas duas técnicas — resposta a palavras-chaves e alteração de tempos verbais — acompanhadas de alguns outros truques especializados, podem produzir um programa que mantenha uma conversação razoável com você! Existem à disposição programas Eliza em BASIC que têm somente algumas páginas — outro testemunho do desejo dos humanos de darem um pouco de sua inteligência aos computadores! A moral desta história toda é que nem tudo que fala com alguém está entendendo o que é dito!!!

Mais Linguagem nos Programas

Linguagem é uma área um pouco diferente, mas, com um pouco de boa vontade e alguma tentativa e erro, pode-se fazer muito mais do que o que é encontrado na safra atual de programas de aplicação. Uma boa idéia para exercitar estas técnicas e se convencer do que foi dito é dar uma nova olhada em algum programa em operação, considerar as oportunidades de inclusão de entrada e saída sob forma de linguagem nele, e tentar implementar estas idéias. Sem dúvida, o processamento de textos necessário será tedioso, e o programa crescerá bastante, mas é surpreendente o que pode ser conseguido com gramática pragmática e as técnicas tipo Eliza. Após implementadas estas modificações e testado o programa com outros usuários, provavelmente serão encontradas ocasiões em que a sintaxe pragmática falha. Observando estas falhas, será fácil alterar o programa para evitá-las e, repetindo-se este processo algumas vezes, será possível chegar a um programa que entenda um subconjunto limitado, mas adequado, de uma linguagem natural, sem ter que resolver o problema geral das linguagens naturais em AI!

8
Abordagens à inteligência

Uma das características mais fascinantes da AI é a quantidade de maneiras em que um método, ou um programa, pode ser descrito ou imaginado, o que pode também constituir um problema quando alguém desejar gerar algo realmente novo! É possível considerar um programa que resolva um determinado problema e não ver que está longe de ser algo novo, simplesmente devido à sua descrição ter sido feita em termos pouco familiares. Talvez o tema individual mais importante de todo este livro tenha sido o fato de que nada é muito diferente em inteligência artificial e que seus programas podem ser compreendidos sem nenhuma teoria difícil. Entretanto, esta maneira prática de ver as coisas, até um certo ponto, desconsidera alguns dos problemas filosóficos centrais da AI, pois, embora esteja claro que os computadores podem imitar alguns aspectos do comportamento inteligente, permanecem abertas ao debate questões tais como "pode um computador ser inteligente?" ou "pode um computador ter uma personalidade?"

Assim, este último capítulo amplia a discussão, adicionando aos aspectos práticos da AI de hoje alguns de seus pontos mais filosóficos. O leitor pode achar que é um pouco tarde para considerar, no último capítulo, se AI conseguirá, enfim, atingir sua meta final, mas, antes de experimentar alguns dos métodos usados atualmente por AI e compreendido como funcionam e são simples, não poderia apreciar a grande distância que separa a situação atual da meta final! Entretanto, antes de atacar estas questões mais amplas, vale a pena examinar algumas das maneiras alternativas e auxiliares para o estudo da inteligência.

O Enfoque Biológico

Os capítulos anteriores deste livro tenderam a considerar AI como um ramo da engenharia, mas não é absolutamente óbvio que isto seja o melhor a fazer, pois os únicos bons exemplos disponíveis de sistemas inteligentes funcionando são de natureza biológica. Existe um outro enfoque para a AI mais relacionado com a biologia, e o único problema com este enfoque é que nunca foi reunido sob o título de uma única disciplina. Alguns dos que trabalham nesta área concentram-se no estudo da maneira de operar do cérebro humano, ou de algum outro animal, e classificam seu trabalho como neurofisiologia, se envolver o exame da estrutura física do cérebro, ou como psicologia, se não for este o caso.

Muito do que é tratado nestes dois campos é relevante para AI e pode fornecer material de suporte, mas raramente será de uso direto. A dificuldade é que a neurofisiologia dá informações, que, pelo menos atualmente, explicam em grande detalhe como funcionam pequenas coleções de neurônios e toda vez que se tenta usar estas informações, como uma maneira de implementar sistemas de AI, sempre se conclui que há maneiras melhores e mais simples de obter os mesmos resultados. Por exemplo, sabe-se da neurofisiologia que existem grupos de neurônios nos cérebros dos gatos que detectam pequenos segmentos de linhas e a maneira como isto é conseguido é interessante. No entanto, é possível construir detectores de linhas igualmente eficientes sem recorrer a neurônios artificiais. Em outras palavras, a neurofisiologia mostra como conseguir coisas simples usando uma tecnologia completamente diferente! Da mesma maneira, a psicologia dá informações excessivamente gerais sobre a maneira que os animais e os humanos se comportam, sendo muito raro que apareça nesta área alguma descrição de *como* alguma coisa funciona com detalhes suficientes para ser de utilidade em AI.

É importante, entretano, compreender que o fato de que a neurofisiologia e a psicologia não tinham dado ainda nenhuma ajuda real à AI não é razão para concluir que nunca

darão! É essencial continuar estudando e aprendendo a partir das implementações biológicas da inteligência que nos rodeiam.

Sistemas Cibernéticos

A cibernética é uma disciplina que tenta estudar os sistemas biológicos do ponto de vista da engenharia, sendo efetivamente difícil definir o enfoque cibernético mais geral, mas seus praticantes tendem a adotar a atitude de que não há, na realidade, diferença entre as máquinas implementadas na tecnologia "mole" das células etc. . . e as implementadas na tecnologia "dura" do silício, sendo mais importante o fato de que todas são máquinas. Esta idéia não é dada mais do que um ponto de vista filosófico, mas tende a concentrar as atenções nos princípios subjacentes e não nos detalhes de implementação. A cibernética começou pelo exame dos mecanismos que controlam os aspectos mais simples dos sistemas vivos. Por exemplo, de alguma maneira, o corpo humano consegue manter uma temperatura bastante estável, mesmo com a temperatura externa flutuando muito. Isto claramente não é um comportamento inteligente, mas, ao tempo em que a cibernética estava começando, os princípios que regem tal regulação eram muito mal compreendidos. Atualmente o princípio fundamental da auto-regulação — a retroalimentação — é muito bem conhecido, e muito importante, em muitos campos da engenharia. A retroalimentação é um fundamento sem o qual as formas mais interessantes e sofisticadas do comportamento inteligente não existiriam. Por exemplo, para pegar alguma coisa que está sobre uma mesa os músculos do braço são controlados por impulsos nervosos do cérebro e a estratégia mais simples é, de alguma maneira, "medir" onde está o objeto, determinar quais impulsos mandar para o braço, enviá-los e esperar até que o braço chegue ao seu destino. Esta é, na realidade, a estratégia usada em muitos robôs mecânicos e sua deficiência torna-se óbvia se o objeto sobre a mesa mover-se enquanto o braço estiver em movimento! Para tentar pegar um objeto em movimento é necessário continuar medindo sua posição e comparando-a com a posição atual do braço. O que realmente acontece quando alguém pega um objeto é que a diferença entre as posições da mão e do objeto é usada pelo cérebro para determinar como mover o

braço para reduzi-la. Obviamente, o movimento somente será completado quando esta diferença se anular e esta estratégia prevê qualquer movimento que o objeto possa fazer. O mecanismo básico envolvido no ato de pegar um objeto pode ser encontrado como parte de muitas outras funções em um sistema vivo, e é chamado de "controle retroalimentado" Sua essência é o modo como o sistema avalia uma diferença, ou um erro, e sua tentativa de anulá-lo.

Embora a retroalimentação seja uma idéia simples, sua implementação prática pode tornar-se muito complicada. Atualmente, grande parte da cibernética está envolvida nas implicações maiores da retroalimentação e em outros aspectos dos sistemas auto-regulados. Por exemplo, o que caracteriza um sistema que pode organizar-se, fazer sua própria manutenção corretiva e reproduzir-se? A questão de se o sucesso do controle retroalimentado como uma explicação da auto-regulação poderá ser ampliado ou substituído por uma nova teoria é algo que somente o tempo dirá. Nos primórdios da AI a ênfase maior era em *hardware*, devido à suposição de que inteligência exigiria novos tipos de máquinas. Hoje em dia, reconhece-se que novas máquinas serão necessárias apenas para fazer os programas funcionarem mais rapidamente, não tendo havido, até o momento, nenhuma grande criação de novas máquinas que introduzissem algo novo em AI. A regra parece ser que o que pode ser feito pode ser implementado tanto em *software* como em *hardware*, sendo este último mais caro, mas também mais eficiente. Entretanto, isto permite também concluir que a incorporação de muitos dos princípios mais simples em *hardware* permitirá aos programadores concentrarem-se nos problemas reais. Por exemplo, é muito mais fácil programar um robô que tenha incorporado em *hardware* a coordenação de seus movimentos usando controle retroalimentado.

Inteligente ou somente esperto?

A linha que separa uma programação esperta da inteligência artificial é muito tênue. Na verdade, pode ser até que não existam programas realmente inteligentes, mas somente programas cada vez mais espertos. Por exemplo, o Capítulo 7 descreveu um programa que pode manter uma conversa limitada,

cujas principais limitações estavam relacionadas com o método bastante grosseiro usado. Detectando palavras-chave, o programa Eliza pode oferecer sentenças em respostas que são relevantes, mesmo que sem sentido. Por exemplo, em resposta a uma sentença usando um "por quê", o programa poderia dizer algo do tipo "algumas questões são difíceis de responder", uma resposta vaga e enigmática que serve para quase toda situação! Para se convencer disto, experimente durante algum tempo responder "certas questões são difíceis de responder" a qualquer pergunta que lhe fizerem contendo um "por quê". Aperfeiçoe seu programa incluindo um sorriso ou um franzir de cenho, dependendo de a pergunta ser leve ou séria! Exceto pelo fato de esta experiência ser um pouco chata, você verá que esta resposta automática é bem aceita pela maioria das pessoas.

Um ponto importante a observar é que a aplicação de algumas regras muito simples dá a impressão de inteligência. Os seres humanos são inteligentes e, se sua expectativa for elevada a um nível adequado, tentarão interpretar qualquer coisa como indício de inteligência em alguma outra pessoa ou coisa. No caso dos computadores, a expectativa do público foi elevada até quase o teto, de maneira que é muito fácil convencer um observador inocente de que algum programa trivial é a quintessência da inteligência do universo.

Não constitui novidade o fato de se atribuir inteligência humana e até personalidade humana às máquinas. Na época das máquinas a vapor, aquelas monstruosas quantidades de bronze e ferro eram freqüentemente chamadas de "ela" e recebiam até nomes próprios! Os humanos têm uma grande tendência a atribuir, mesmo às máquinas mais simples, um pouco de sua inteligência ou de sua personalidade. O programa Eliza apareceu na aurora da computação, assim como a AI, e, naquela época, computadores eram uma novidade tão recente que muitas pessoas ficaram tão impressionadas com seu desempenho que mantiveram longas e profundas conversações com o programa. Eliza foi tomado tão a sério que certos psiquiatras procuraram usá-lo com seus pacientes, sendo que alguns destes preferiram o programa aos médicos! E a situação não mudou muito desde aquela época, pois a grande sofisticação atual

não levou o programa Eliza a ser considerado algo antigo e pouco impressionante, como merece, mas ainda é exibido como exemplo de inteligência computacional ao invés de um monumento histórico. Muitas pessoas do público (e mesmo muitos cientistas de computação) ainda são enganados por ele!

A tendência a acreditar na inteligência dos computadores tem dois importantes aspectos. Primeiro, como já foi várias vezes mencionado, significa que é possível alcançar resultados práticos sem muito esforço, simplesmente emprestando um pouco da inteligência do usuário. Em segundo lugar, é uma advertência de que não é razoável acreditar muito rapidamente em tudo que se vê!

O teste de Turing

A maneira tradicional de se precaver contra a tendência a acreditar que um programa simplesmente esperto já tenha atingido o que se deve chamar de inteligência, é o chamado "teste de Turing", que recebeu este nome em homenagem a Alan Turing, seu inventor. A idéia básica do teste é que qualquer programa que se defina como inteligente deve ser comparado com um humano, a única forma de inteligência conhecida. O teste consiste em colocar ambos, o programa e o humano, em salas separadas e fechadas, permitindo a outros humanos comunicarem-se com eles por alguma via que não identifique o destinatário. Se estes observadores externos, após várias experiências, não puderem distinguir qual é o humano e qual é o programa, então este último merece o nome de inteligente para quaisquer finalidades práticas! Este teste simples tem, por sua vez, muitos problemas, tais como, por exemplo, deve o humano tentar esconder ou evidenciar sua condição?, mas é muito atraente como uma maneira de descrever o que se deve esperar de um programa inteligente. O maior problema, entretanto, é que o teste de Turing é um teste tipo "caixa preta", já que os observadores não precisam preocupar-se com a maneira pela qual as coisas são realizadas dentro da caixa, mas somente verificar se suas saídas são apropriadas. Isto constitui um problema, já que é possível que um programa passe no teste de Turing e receba o rótulo de "inteligente", enquanto que, para o cientista de computação que o criou, pareça muito mais sim-

ples do que a verdadeira inteligência humana na sua maneira de operar. Alguns programas existentes já passam pelo teste de Turing, quando este é limitado a questões restritas a uma estreita gama de assuntos. Por exemplo, os programas que jogam xadrez já são tão bons que somente um mestre pode dizer a diferença entre um programa e um jogador humano. Neste sentido, o programa de xadrez é inteligente no que respeita a questões sobre xadrez, mas, como já foi visto, o programa usa métodos que provavelmente não são os mesmos usados pelos humanos. Um programa de xadrez pesquisa uma árvore de movimentos, usa funções de avaliação exatas, aplica regras heurísticas e tem registradas as aberturas e os finais com mais precisão do que qualquer jogador humano. Em outras palavras, muito do seu desempenho vem de uma programação muito hábil combinada com a grande velocidade de cálculo do computador. Outro problema com o teste de Turing é que, em certo sentido, ele é muito fácil de passar. Por exemplo, colocando-se o programa Eliza em uma caixa preta é fácil encontrar alguém que se convença de que ele é inteligente. Isto dá ao programa Eliza o direito de ser classificado de inteligente? Claro que não, mas, em um certo sentido, ele passou no teste de Turing!

Uma boa modificação para o teste de Turing, permitindo-lhe levar em consideração as habilidades especiais de alguns programas e as díspares habilidades dos humanos, consistiria em dar um tipo de grau de QI, medido pela relação entre as pessoas que acreditam que o programa é inteligente e as que não acreditam. Assim, se um programa de xadrez convencer 80% das pessoas de que é inteligente, esta será uma medida grosseira de seu QI. Para tornar esta medida mais perfeita, seria necessário levar em consideração os QIs das pessoas que o programa convenceu de que é inteligente, ou seja:

> QI específico do programa = média dos QIs específicos de todas as pessoas que decidiram que o programa passa no teste de Turing.

A vantagem desta definição é que ela permite dizer quando um programa alcançou o nível do humano médio que executa a tarefa específica, eliminando a ênfase atual de que o programa deve "ganhar dos melhores". Além disto, transforma o teste de Turing em uma medida (a medida de Turing?) da inteligência do programa, ao invés de uma prova tipo tudo ou nada. É claro que tudo isto dependerá de como serão realizadas as medidas dos QIs dos humanos, mas este é um problema que pertence mais à área da psicologia (da psicométrica, para ser mais preciso).

É claro que a inteligência humana é diferente de qualquer das habilidades medidas pelos testes de QI, pelo fato de ser geral, mas o que isto significa? No caso do programa de xadrez, é fácil ver que um programa que recebesse um elevado QI para xadrez poderia ter um QI de aproximadamente zero em todos os outros assuntos, o que não faria nenhuma diferença! Por outro lado, um jogador de xadrez humano receberia graus não-nulos para o QI em muitas outras habilidades, o que seria uma clara indicação de que o QI geral dos humanos difere do QI extremamente específico dos programas de computador. A meta de longo prazo da AI é produzir um programa que simule uma parte suficientemente grande do comportamente humano para obter graus de QI significativamente maiores do que zero em uma gama ilimitada de testes específicos de QI.

É a inteligência computável?

A meta de longo prazo da AI é algo ainda sujeito a controvérsias. Alguns seguem o argumento cibernético de que os humanos são simplesmente casos especiais de um princípio mais geral, subjacente a todas as máquinas, ou seja, os humanos não são muito diferentes de suas criações. Este argumento tem uma simplicidade que o torna muito atraente, mas tem também uma longa história sem que alguém tenha conseguido apresentar evidência conclusiva de que está errado! Na verdade, à medida que a biologia progride, cada vez maior parte dos sistemas humanos e animais em geral é explicada em uma linguagem mecanística, semelhante à usada para as máquinas criadas pelo homem. Já houve tempo em que se pensava que a vida continha alguma qualidade mágica que falta aos objetos inanimados — uma espécie de força diferente, não fazendo parte do restante do mundo físico. Atualmente, a vida é descrita em termos de reações químicas que reduzem mesmo o corpo

humano a uma espécie de mecanismo, tendo muito em comum com a mais simples das máquinas a vapor! Este é somente outro aspecto da tendência geral de tornar o sistema humano cada vez mais semelhante do resto do universo. Antigamente, havia oposição à simples idéia de que os humanos tinham algo em comum com os animais, e muito menos com o mundo inanimado das máquinas! Depois, com a aceitação da evolução de Darwin, a distinção entre os humanos e os animais foi enfraquecida e, à medida que a ciência evolui, a distinção entre o animal físico e a máquina física torna-se cada vez menos clara — sendo mais uma diferença na tecnologia usada para implementar os mesmos princípios. A tentativa de remover a última distinção ainda existente entre as máquinas e os humanos, incluída na meta da AI, de produzir computadores inteligentes, pode ser vista como mais um estágio nesta progressão. Entretanto, o progresso constante da explicação mecanística não é uma garantia de que haverá, no futuro, uma redução bem-sucedida do mundo a princípios mecanísticos. Excetuando admitir a existência de alguma coisa mágica que os humanos tenham e que crie inteligência, é difícil imaginar alguma outra coisa que impeça a produção de inteligência nas máquinas.

Há uma crença generalizada de que, dado tempo suficiente e grande sofisticação, uma máquina tipo computador poderia resolver qualquer problema. Este é um dos pontos falsos do argumento freqüentemente repetido de que os computadores serão um dia tão inteligentes quanto os humanos. A falha neste argumento é que muitos problemas conhecidos não são computáveis! Os argumentos exatos que mostram que existem problemas não computáveis envolvem muita matemática avançada e os próprios problemas são bastante esotéricos e difíceis de descrever. Talvez o mais simples e, simultaneamente, o mais relevante deles é que é impossível escrever um programa que determine se outro programa irá parar. Em outras palavras, é impossível escrever um programa que aceite como entrada outro programa e imprima uma mensagem indicando se ele irá eventualmente parar ou se irá rodar para sempre. Isto significa que o problema da "parada" do programa não é computável. A descoberta da existência de tais problemas não computáveis foi um grande choque para a comunidade científica em geral, e houve muitas tentativas de usá-lo na discussão da inteligência humana. A pergunta real é "terão os humanos algo que não pode ser programado em um computador que impeça os computadores de chegarem a possuir inteligência?" Em outras palavras, "será a inteligência computável?"

O progresso contínuo da redução de todas as coisas, tanto animais como humanas, à máquina é, num certo sentido, encorajadora para a crença de que a inteligência seja efetivamente computável. Entretanto, este constante progresso encontra um tipo de descontinuidade quando se aproxima da inteligência. A importante diferença é que tanto a inteligência como os computadores são ambos produtos do cérebro humano e que, enquanto este pode entender o funcionamento do corpo e de outras máquinas, a idéia de que possa entender a si próprio é um pouco mais surpreendente! Na verdade, há algo parecido com o problema da parada do programa — ou seja, um programa examinando outros programas — em um cérebro que examina outros cérebros e já se sabe que o problema da parada não é computável!

Não estou sugerindo, com isto, que a inteligência não seja computável. Ao contrário, a situação atual na área é a de crença em que o constante progresso da AI levará, finalmente, à criação da máquina inteligente.

Será AI Diferente?

Há aqueles que acham que AI não é realmente diferente da programação normal e da ciência de computação. Isto é realmente verdade devido a que, até agora, os especialistas em AI não conseguiram produzir programas realmente inteligentes. Nenhum programa existente de AI usa algo tão difícil que não seja prontamente reconhecido e compreendido por um bom programador. A dificuldade consiste em que é difícil ver como escrever um programa que possa ser chamado de inteligente, se quem o faz conhece em grande detalhe tudo o que ele faz! Neste sentido, AI é um assunto que busca alcançar um objetivo mutável, pois cada vez que se consegue escrever um programa que execute algo que se pensava que exigia inteligência, todos os seus detalhes de funcionamento ficam aparentes e a tarefa acaba mais bem classificada na área da automação. Um programa real-

mente inteligente deveria ser tal que conseguisse enganar não somente o público em geral, tal como o Eliza, mas também seus criadores, o que é difícil de imaginar.

Mesmo que ainda não haja a mágica da máquina inteligente real para distinguir AI das outras áreas da programação, já existe uma coleção de métodos próprios de AI. Assim, a idéia de um algoritmo aplicado de forma um pouco vaga como uma heurística é característica dos programas de AI. A estrutura em árvore surge tão freqüentemente e em uma gama tão ampla de problemas, que pode ser dita a estrutura de dados fundamental da AI. Na realidade, pode-se dizer que a árvore está para AI assim como o *array* está para a programação matemática. Muito da teoria da AI vem de outras disciplinas — lógica, filosofia, probabilidade etc... — mas estas aparecem com menor freqüência. Mesmo assim, o programador em AI deve ser muito versátil.

E o que Virá a Seguir?

Pela primeira vez AI está sendo seriamente considerada como um ramo da ciência dos computadores e, até um certo ponto, a razão para isto é que os milhões de computadores pessoais já vendidos e os milhões de máquinas mais poderosas que virão a seguir terão que usar AI para serem bem e amplamente utilizados. Não há maneira de aumentar o número de programadores na mesma razão do aumento do número de computadores e de suas aplicações. Sempre houve falta de recursos humanos bem capacitados em computação e esta situação somente tende a se agravar!

Se, eventualmente, os computadores se tornarem úteis a todos, e não somente uma exclusividade dos cientistas, técnicos e programadores, deverão transformar-se em recursos dos quais todos possam obter alguma coisa. Sempre será verdade que algumas pessoas conseguirão tirar mais proveito dos computadores que outras — é provável que continuem existindo pesquisadores de AI no futuro! — mas o objetivo é baixar o nível do conhecimento necessário para usar um computador. Para isto será importante concentrar parte do desenvolvimento de AI na produção de sistemas flexíveis que possam interagir facilmente com humanos para fornecer e registrar conhecimento.

O computador como um amplificador de inteligência é um conceito atrativo que ainda está longe de ser implementado. Atualmente, a maior parte do trabalho conjunto entre homens e máquinas é feito nos termos das máquinas! É possível ver o constante desenvolvimento do *software* em termos de sua adaptabilidade ao usuário. Os *software* de "primeira geração" procuravam somente executar a tarefa pedida, não prestando nenhuma atenção à conveniência dos usuários. O *software* de "segunda geração" tende a tratar as mesmas aplicações com as mesmas técnicas, mas adicionando características que o tornem "amigável ao usuário". Este termo "amigável ao usuário" (do inglês *user-friendly*) é difícil de definir, mas pode ser anterpretado como significando que o *software* informara ao usuário sobre quaisquer erros que tenha cometido, dando-lhe nova chance para corrigi-los (ao contrário dos *softwares* de primeira geração, que imprimiam uma mensagem de erro e simplesmente interrompiam o processamento!). *Softwares* de "terceira geração" estão começando a aparecer, sendo caracterizados pela hipótese de quaisquer mal-entendidos ou erros são culpa do *solftware* e não do usuário! Por exemplo, um programa de banco de dados de primeira geração esperaria solicitações dentro de um formato fixo, adequado para o processamento interno, sendo que qualquer solicitação fora do formato seria simplesmente rejeitada. Já um programa de banco de dados de segunda geração seria um pouco melhor, particularmente no fato de usar formatos para solicitações mais de acordo com as necessidades dos usuários, e, se detectasse algum desvio do formato, solicitaria uma correção do usuário. Um programa de banco de dados de terceira geração não forçaria o usuário a introduzir suas solicitações em nenhum formato que não a linguagem mais natural possível e, qualquer mal-entendido que resultasse, seria debitado a falhas do programa, que procuraria não repetir o erro, buscando aprender a entender o significado deste novo tipo de solicitação do usuário, ao invés de forçar o usuário a aprender a apresentar a solicitação de outra forma.

A meta final da AI é criar inteligência de tipo humano em uma máquina não-humana. Se isto pode ser atingido num período razoável de tempo, ou se será eventualmente conseguido, não é uma questão que altere a importância prática de produzir

programas que levem cada vez mais adiante no caminho desta meta. A nova geração de programas que AI está produzindo deve constituir o início de uma modificação gradual da maneira atual das máquinas procederem para uma maneira mais flexível e inteligente.

Leitura adicional

A literatura de AI pode ser dividida em duas categorias: discussões teóricas de como as coisas poderiam ser feitas e descrições de como as coisas foram efetivamente feitas. Embora as discussões teóricas sejam algumas vezes úteis, não estando limitadas pelas restrições do mundo real, tendem a escapar para o reino da filosofia! Os melhores livros de AI procuram dizer algo prático, baseado na teoria!

Existe um grande número de textos introdutórios sobre AI. Meus favoritos são:

Artificial Intelligence, de P. H. Winston, Addison-Wesley, 1977
Uma descrição muito simples de uma ampla gama de assuntos, inclusive uma introdução à principal linguagem de programação de AI, LISP.

Artificial Intelligence and Natural Man, de M. Boden, Harvester Press, 1977
Uma boa introdução não-técnica à AI.

Há um enorme número de livros sobre tópicos especiais, não sendo possível dar uma lista de tudo o que vale a pena ler. Alguns que eu achei muito úteis são:

Build Your own Expert System, de C. Naylor, Sigma Technical Press, 1983
Este livro usa decisão Bayesiana como uma introdução aos sistemas especialistas. É interessante consultá-lo para um enfoque diferente do que foi dado aqui.

Expert Systems in the Micro Electronic Age, editado por D. Michie, Edinburgh University Press, 1979
O melhor guia aos sistemas especialistas práticos, muito fácil de ler, desde que se tenha dominado o essencial.

Brains, Machines and Mathematics, de M. A. Arbib, McGraw-Hill, 1964
Um guia bom, mas um pouco matemático demais, à cibernética e ao enfoque biológico em geral.

Computer Gamesmanship, de D. Levy, Century Publishing Co., 1983

Índice analítico

Ajuste ao gabarito, 116
Algoritmo, 23
Análise gramatical, 137
Aprendizado, 112
Armazenamento conceitual, 99
Armazenamento relacional, 96
Árvore de movimentos, 36

BASIC, 19

Característica, 115, 122
Cibernética, 152
Computabilidade, 19, 157
Correlação cruzada, 121

Endereço, 93
Espaço das características, 123

Função de avaliação, 47

Heurística, 21, 33, 43, 58

Inteligência, 14
Inteligência auxiliada por computador, 13

Jogos de duas pessoas, 45

Linguagem, 133
LISP, 19
Lógica nebulosa, 79

Memória, 90
Memória associativa, 95
Memória computacional, 90
Memória humana, 91
Microsoft BASIC, 19
Minimax, 53

Perceptron, 128
Pesquisa de duas camadas, 38, 53
Probabilidade, 73
Probabilidade condicional, 74
Programa Aardvark, 64
Programa de conversa de computador, 140
Programa do banco de dados conceitual, 108
Programa do jogo da velha, 45, 82
Programa do jogo dos quadradinhos, 24, 40
Programa Eliza, 147, 154
Prolog, 19

RAM, 18
Reconhecimento da fala, 16
Reconhecimento de padrões, 111

Regra de produção, 137
Resolução de problemas, 17, 22, 62

Semântica, 134, 143
Significado, 143
Sintaxe, 134
Sintaxe pragmática, 146
Sistema especialista, 59

Teorema de Bayes, 76
Teste de Turing, 155

Visão computacional, 15

Este livro foi impresso nas oficinas gráficas da
Editora Vozes Ltda.,
Rua Frei Luís, 100 — Petrópolis, RJ,
com filmes e papel fornecidos pelo editor.